WEB BASED STOCK FORECASTING

SOFTWARE ENGINEERING WEB APPLICATIONS

PROJECT REPORT

GROUP 1 Date of submission: 5/7/08

Parvathy Sreekumar, Rahul Pandey, Raghavendra Sidhanti, Rahul Suryawanshi, Sreejith Sreedharan

Group website: software-ece.rutgers.edu/~group801

Contribution Breakdown:

1) User Interface development, Web service Development, all the .NET code: **Parvathy Sreekumar**.

2) Monte Carlo Predictor, Evaluation: **Raghavendra Sidhanti**.

3) User Interface development, System Integration, Report and UML Designs, Predictors Evaluation: **Rahul Pandey**.

4) Data Collection from Yahoo!Finance, Data Base Development(Using PHP script), Integration between prediction modules and web service: **Sreejith Sreedharan.**

5) SVM Pattern Predictor, Evaluation: **Rahul Suryawanshi.** .

Table of Contents

1. Introduction

Our project is aimed at providing investors (users) with advice regarding making investments in stocks by predicting the future price value based on historic price. This will provide a guideline for short term and long term investors to make decisions while investing in stock market. Our project has two predicting algorithms which is the important feature of our web site.

The user interface of the web site is user-friendly and simple. We have implemented complex and smart algorithms to perform price predictions. These algorithms run as a back-end task and compute the prediction values for the various stocks.

This report describes the overall structure, behaviors and interactions of various modules of our system. It also contains the details of the algorithm, prediction modules, databases and web services used in successfully implementing our project. The figure 1.1 shows the overall system block diagram, different modules which are numbered from 1 through 6. In the sections below we try to explain first the structure of modules then behavior and in the end its interaction with other module and in the end how this module help us in achieving the final result.

The mail purpose of the project was to implement a system which will make use of web-services and facilitate the future stock prediction. For predicting the stock's future we implemented two algorithms one is predicting future value and value-at-risk estimation based on the Monte Carlo method, and the other is SVM pattern predictor. The web services are implemented using .Net framework. We have are two predicting modules that generate the desired results for our client. The database holds the updated Yahoo! Finance data, as well as the results generated by the predictors.

The Fig 1.1 illustrates an abstraction of the whole system; the user interacts only with the user interface which can be seen as the controller in the model-view-controller design pattern. The UI then invokes the web-services which get the data from database and shows the result back to the user. The predictors work in parallel and update the calculated results in the database. The detailed explanation of every module is given in the next few sections.

.

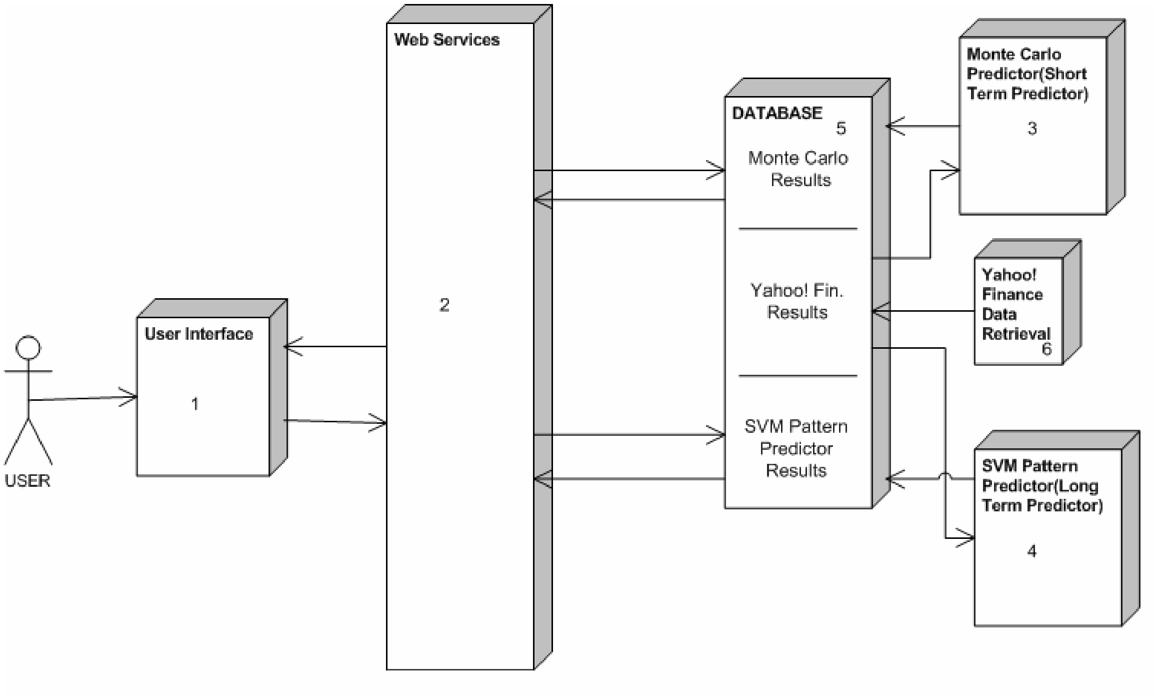

Web Services
2
DATABASE
5
Monte Carlo Results
Yahoo! Fin. Results
SVM Pattern Predictor Results
Monte Carlo Predictor(Short Term Predictor)
3
Yahoo! Finance Data Retrieval
6
SVM Pattern Predictor(Long Term Predictor)
4
User Interface
1
USER

Fig 1.1: System block model of Web based Stock Forecaster. It illustrates 1) User Interface 2)Web Service 3)Monte-Carlo Predictor 4)SVM Pattern Predictor 5)Database 6)Yahoo! Finance data retrieval.

It is clearly seen from the diagram that how these 5 modules interacts with each other and are integrated seamlessly. We design in such a way so that our prediction algorithm can work separately. Since the Monte-Carlo takes approximately three minutes(based on our system specification stated below) to compute the future stock value, its decision to buy, sell, hold and value at risk for all the forty eight stocks we are tracking it was difficult for us to call/invoke the prediction program by other modules(like web-service would have directly called it), but we are running this module and the SVM pattern predictor module separately in a delay of twenty seconds once data entry of stocks are done in the predicted table (st_val- is the table for Monte Carlo's predicted vale, lt_val- is the table for SVM pattern predictors values) so that there would not be any overlapping of processing time and request time. Also this architecture allow us implement the system easily since predicting modules become separate entity and support our web-service separately, which can give service to third party without any overhead of prediction module capability. Unbiased prediction from other web service can be carried out pretty easily after making use of our website, for both the short term investors and long term investors.

<u>System Specification's</u>:

Main server: Sun Microsystems sun4u Sun Ultra 80 UPA/PCI (2 X UltraSPARC-II CPU Speed: 450MHz),GCC version 3.4.6, Matlab Version :7.5.0.338 (R2007b), PHP Version: 5.0.

1.1 System Design

Actors and Goals: The following table describes the actors and goals of each of them and the use case these goals are represented by.

Actors	Actor's Goal	Use Case Name
User	Get short term stock prediction	(UC1) GetShortTermPrediction
User	Get long term stock prediction	(UC2) GetLongTermPrediction
User	Retrieve predicted value from database and provide service to its clients	(UC3)Web Service
Database	Predict short term stock price, decision(Buy/Sell/Hold), Value at risk	(UC4)Monte-Carlo Predictor
Database	Predict long term stock future	(UC 5) SVM Pattern Predictor
Database	To get the current and historical value of the stocks	(UC 6) Data Retrieval

Table:1.1 Actors, goals and the associated use for the Web based stock forecaster.

To start with our user who is the initiating actor has two goals either the actor is a short term investor looking for predicted stock value for near future (we update our advice every twenty minutes) invokes either GetShortTermPrediction or GetLongTermPrediction use case from the user interface. This in turn invokes the web service use case. Now the purpose of Web Service is to give its service to the client/third party. So the user invoking this use case can either be an individual client or any third party website, here it is our individual client using our web site. Monte Carlo and SVM Pattern use cases are running separately making use of the Data Retrieval use case which populates the year long historical data of the forty eight companies. In return the predictor use case stores its results to the data base. The web service takes that result and shows in the user interface as a return to GetShortPrediction or GetLongTermPrediction call.

1.2 Use Cases:

A use case is a description of what happens when an external entity, known as actor, disturbs our system form its “stationary state” as the system goes through its motions until it reaches a new stationary state. It is important to keep in mind that the system is *reactive*, not proactive; that is, if left undisturbed, the system would remain forever in the equilibrium state. An actor can be a person or another system which interacts with our system. The list of use cases which we form our system is described in Table 1. There are eight main use cases. Fig 1.2 shows the use- case diagram for player management in the Web based stock predictor.

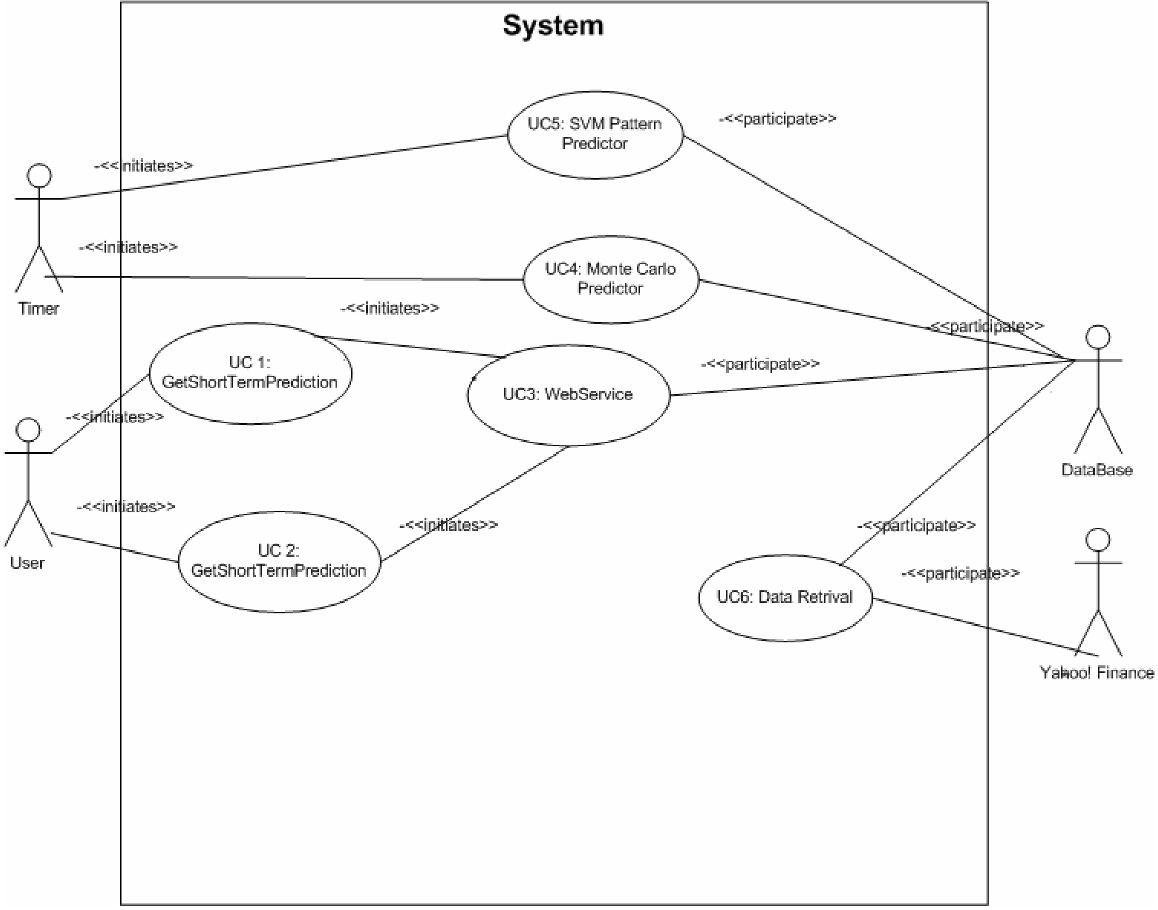


Fig 1.2 Use Case Diagram of Web Based Stock Forecaster.

There are four actors- User, Database, Timer and Yahoo! Finance. The user’s goal is to get the predictions in form of either long term prediction (SVM Pattern predictor) or short term prediction (Monte-Carlo predictor). The user can obtain this by using our web-service which invokes the updated value. The short term predictor (UC 4) which is initiated by timer every thirty minutes and it updates the predicted values in the database, while the long term predictor (UC 5) which is also initiated by the times, updates the value ones a day. The initial values that are required for predicting the future stock are the historical data, which we obtain from Yahoo! Finance by the use of data retrieval .

1.3 Some Use Case Description’s

UC - 4 Monte-Carlo Predictor

Initiating Actor: Timer.

Actors Goal: Predict short term stock price, decision (Buy/Sell/Hold), Value at risk

Participating Actor: Database

Pre-conditions: Historical data in the database.

Post Condition: Simulates the future stocks value and updates the table ”st_val” in the database(it updates every 30 mins)

Flow of events for the Main Success Scenario

1. Data Retrieval: PHP code retrieves historical stock data from Yahoo! Finance and inserts it in to the database.
2. MC Predictor: Retrieves the historical stock data and predicts the future stock values. Inserts decision, simulated value and Value-at-Risk in to the database.

UC – 5 SVM Pattern Predictor

Initiating actor: Timer.

Actors Goal: Get long term stock prediction.

Participating Actors: Database

Preconditions: Initial data representing patterns and historical stock values.

Post conditions: Predicts the pattern and gives provides the long term decisions.

Flow of events for the Main Success Scenario:

1. Data Retrieval: PHP code retrieves historical stock data from Yahoo! Finance and inserts it in to the database.
2. SVM Predictor: Retrieves the historical stock data and predicts the future stock values. It inserts the pattern and decision in to the database.

2 User Interface

We tried to make our user interface easy to understand for the user and also made it simple by separating the backend complex engines. Our web site is open to all client and we take advantage of our web service to cater future stock prediction. The initial page appears in front of the user is a welcome page giving a brief description of the web site. In this welcome page the user have two options to choose from, one is either user wants to predict for a stock its short term prediction by the use of our Monte-Carlo predictor or want to know about the stock's long term prediction by SVM Pattern predictor. If the user chooses the short term prediction we take it to second page which has the interface for taking service of the Monte Carlo's predicted value. The user is given a choice to pick a company symbol out of forty eight companies we are tracking from a drop down box. To predict all user need to do is to click the predict button. After hitting the predict button our web service is invoked and the updated company stock prediction value is returned to the client. Three decisions are shown here one is the predicted future stock price, our decision to sell, buy, hold stock and value at risk calculation of our prediction engine. If the user wants to take advice for long term investment he/she can go for the long term prediction option. This also has the same simple user interface, user needs to pick the company symbol and simply click on predict button, in return to this the web service response returns our decision to sell, buy, hold along with the pattern our SVM predictor returned.

The fig 2.1 illustrates the sequence diagram for short term prediction showing actions that client/user invokes. After user press the predict button the action is transferred to webservice through ShortTermPrediction function which in return invokes GetShortTermPrediction code web service, the details of these functions and classes are given in the preceding sections. The same sequence of action is carried out for the long term prediction. We have used two different methods for invoking web-service prediction code and server predictor modules i.e ShortTermPrediction( ), LongTermPrediction( ), GetShortTermPrediction( ), GetLongTermPrediction( ), we designed our system this way and used two distinguished methods since the short term predictor returns three values(future stock value, decision, and value-at-risk) and the short term predictor returns two values(predicted pattern and decision). So it was difficult for us to invoke using one method like getPrediction( ).

Fig 2.2 shows a screenshot of our user interface for the short term prediction (Monte Carlo prediction technique). Fig 2.3 shows a screenshot of the long term investment prediction (SVN predictor technique). Fig 2.4 shows a screenshot of the stock history information that we get on clicking "Research" link in the menu bar.

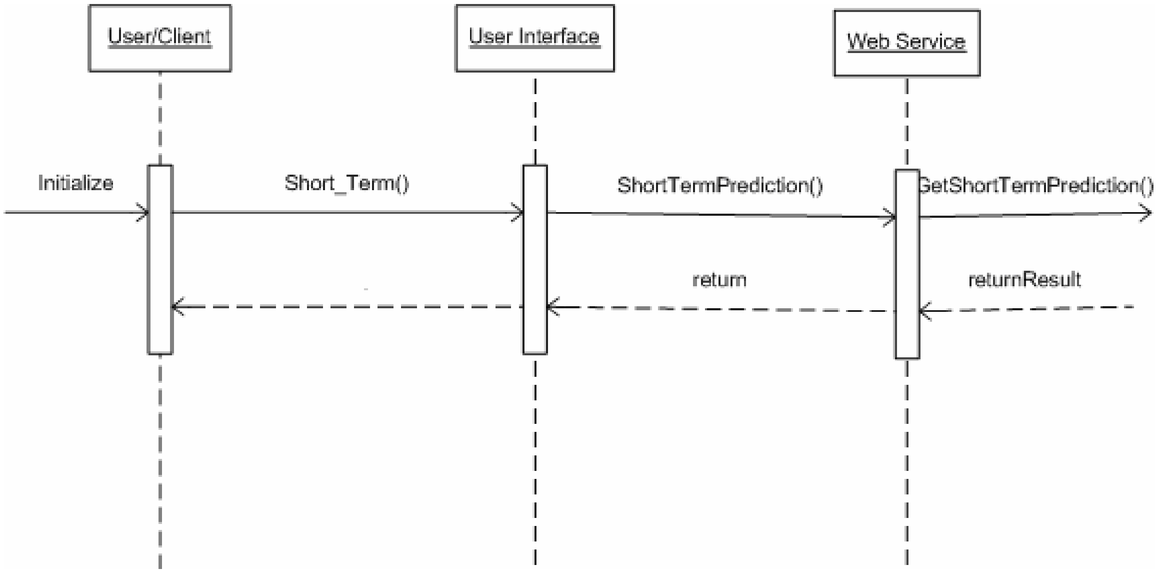
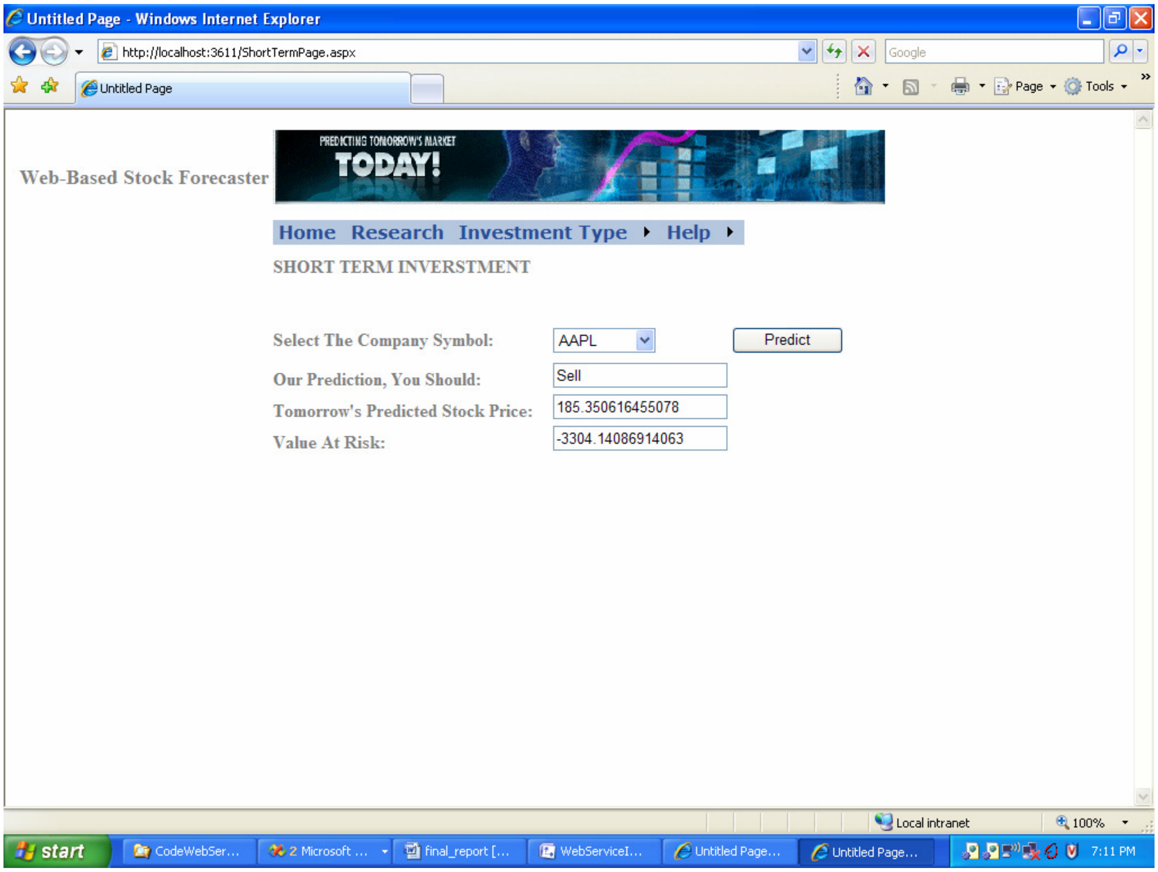


Fig 2.1 Sequence diagram for short term prediction.



Web-Based Stock Forecaster

SHORT TERM INVERSTMENT

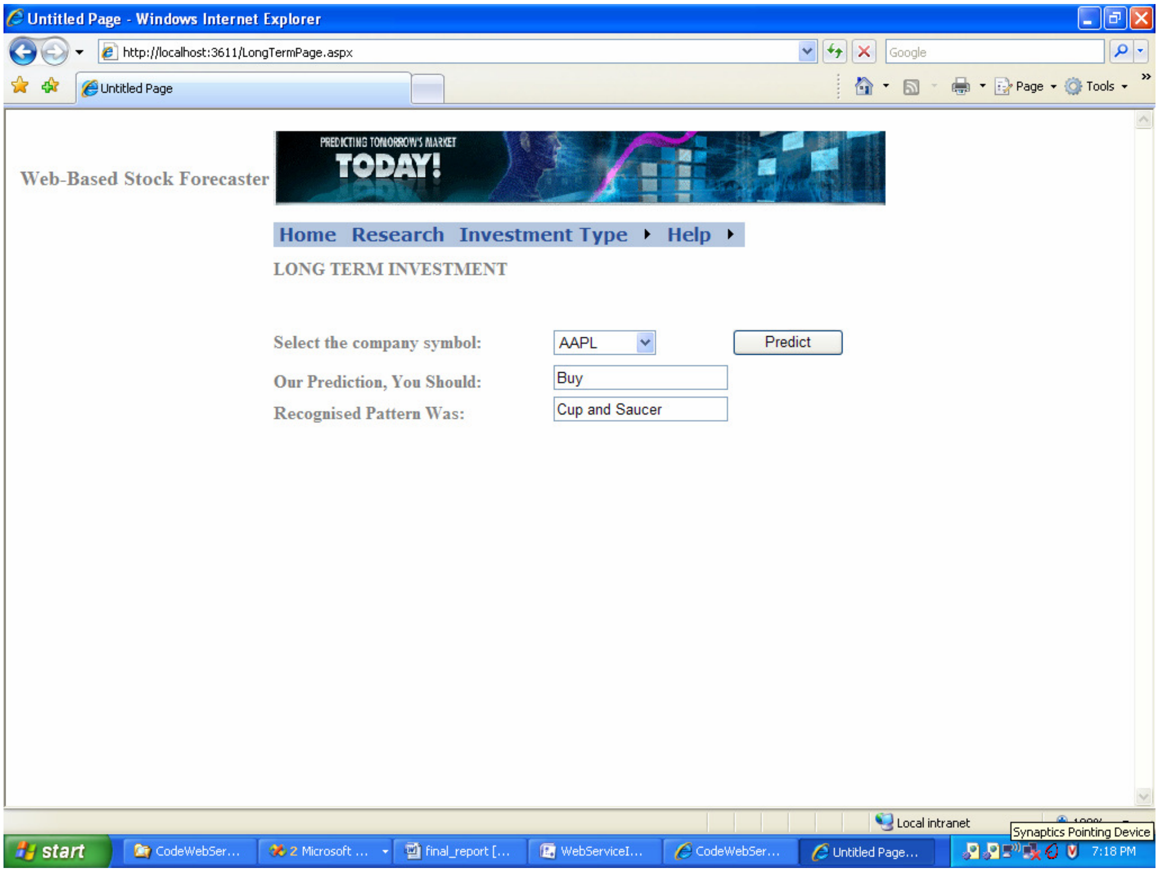
Select The Company Symbol: AAPL

Our Prediction, You Should: Sell

Tomorrow's Predicted Stock Price: 185.350616455078

Value At Risk: -3304.14086914063

Fig 2.2 Screenshot of Short Term Investment Prediction



Web-Based Stock Forecaster

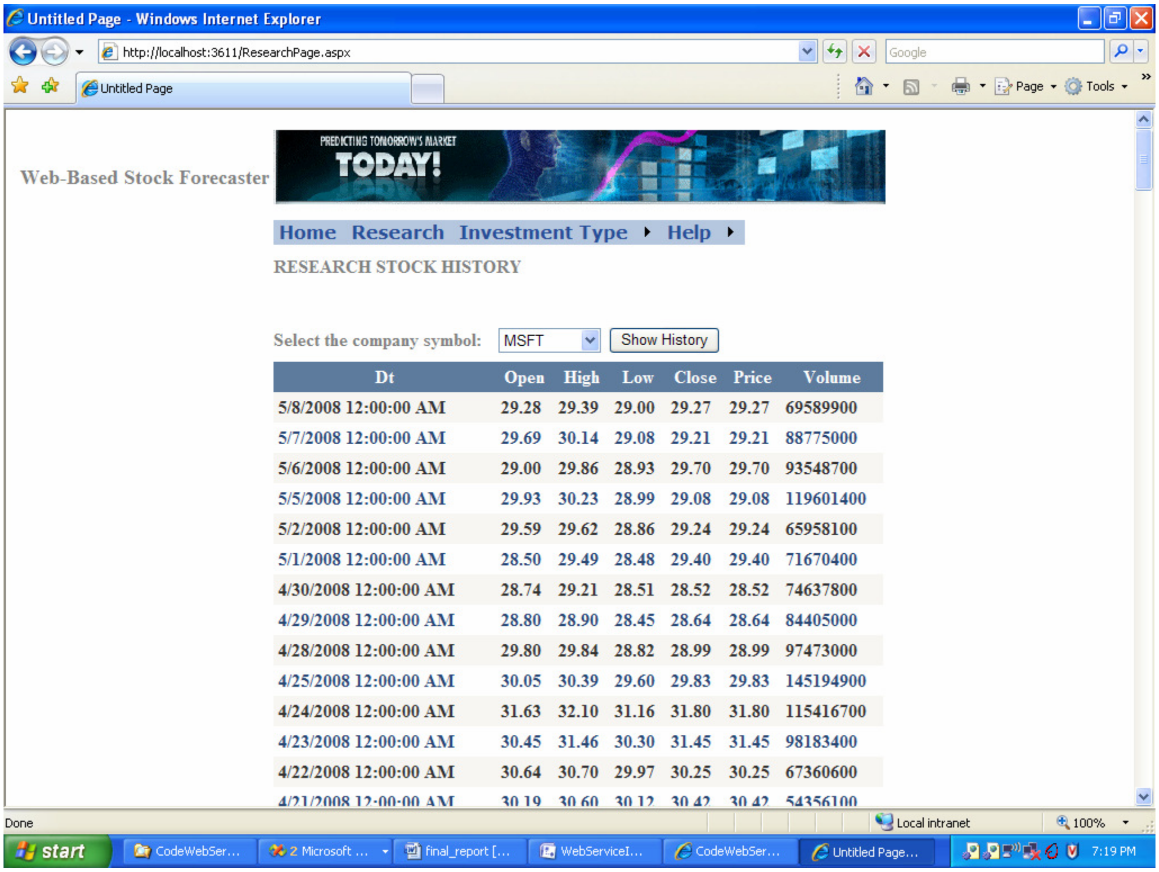
LONG TERM INVESTMENT

Select the company symbol: AAPL

Our Prediction, You Should: Buy

Recognised Pattern Was: Cup and Saucer

Fig 2.3 Screenshot of Long Term Investment Prediction



Web-Based Stock Forecaster

RESEARCH STOCK HISTORY

Select the company symbol: MSFT Show History

Dt	Open	High	Low	Close	Price	Volume
5/8/2008 12:00:00 AM	29.28	29.39	29.00	29.27	29.27	69589900
5/7/2008 12:00:00 AM	29.69	30.14	29.08	29.21	29.21	88775000
5/6/2008 12:00:00 AM	29.00	29.86	28.93	29.70	29.70	93548700
5/5/2008 12:00:00 AM	29.93	30.23	28.99	29.08	29.08	119601400
5/2/2008 12:00:00 AM	29.59	29.62	28.86	29.24	29.24	65958100
5/1/2008 12:00:00 AM	28.50	29.49	28.48	29.40	29.40	71670400
4/30/2008 12:00:00 AM	28.74	29.21	28.51	28.52	28.52	74637800
4/29/2008 12:00:00 AM	28.80	28.90	28.45	28.64	28.64	84405000
4/28/2008 12:00:00 AM	29.80	29.84	28.82	28.99	28.99	97473000
4/25/2008 12:00:00 AM	30.05	30.39	29.60	29.83	29.83	145194900
4/24/2008 12:00:00 AM	31.63	32.10	31.16	31.80	31.80	115416700
4/23/2008 12:00:00 AM	30.45	31.46	30.30	31.45	31.45	98183400
4/22/2008 12:00:00 AM	30.64	30.70	29.97	30.25	30.25	67360600
4/21/2008 12:00:00 AM	30.19	30.60	30.12	30.42	30.42	54356100

Fig 2.4 Screenshot of stock history

3 Web service infrastructure

The .Net framework abstracts most of the internal logic that handles the remote details of method calls over the wire and Visual Studio .Net builds support for Web Services directly into the development environment. A Web Service in .Net consists of a .ASMX page that either contains a class that provides the Web Service functionality or references a specific external class that handles the logic in an external class file. Classes are standard .Net classes and the only difference is that every method that we would like to expose to the Web is prefixed with a [WebMethod] attribute. Once the .ASMX page has been created the Web Service is ready for accessing over the Web.

Behind the scenes there are three major components that make up a Web Service:

1. The Web Service on the Server side
2. The client application calling the Web Service via a Web Reference
3. A WSDL Web Service description that describes the functionality of the Web Service

Our WebService is defined in the file CodeWebService.asmx. The service is implemented just as a class and each method is exposed by using the [WebMethod] attribute to tell the compiler to expose the method. When we create a Web Service we automatically get support for a WSDL (Web Service Description Language) schema that describes the Web Service by accessing the .ASMX page on the fly with a query string of WSDL. The WSDL can be seen on clicking "Service Description" on the web service interface. This is the default interface provided by the .Net infrastructure. A list of web methods is shown on the interface. The SOAP message format is also displayed. The "skeleton" proxies for the server side are created by .Net Framework behind the scene. A WSDL file describes all the methods and method signatures, as well as the namespaces and the handling URL for the Web Service in an XML document. Visual Studio.Net uses the WSDL file to create a Web Reference on the client side from the Web Service. This Web Reference acts as the stub proxy on the client side. The code for this stub proxy resides in Reference.cs file located in ClientService\Web References\CodeWebService (in the project zip folder that will be submitting). It reads the WSDL file and based on the definitions found in the WSDL file creates a proxy class that mimics the interface of the Web Service. The resulting class is actual source code. This proxy class inherits from System.Web.Services.Protocols.SoapHttpClientProtocol class defined in the .NET framework. This base class has the knowledge on how to translate the request from the client to a SOAP message and ship it over the network[22].

The process of connecting a client application in Visual Studio.Net is as follows[24]. We set up a Web Reference, add the Web Reference namespace and then simply call the methods of the Web Service. Behind the scenes the method call actually calls a proxy object, which invokes the remote Web Service. The proxy base class performs the SOAP call over the wire and the proxy class calls work methods in the base class (client side). The proxy reads the WSDL file to verify that the method signature and type information is correct and up to date, then creates the SOAP envelope to send to the Web Service for processing. The proxy makes the HTTP call and passes the SOAP packet off to the Web server. On the server side the .Net framework handler through ASP.NET and the .ASMX Web Service extension picks up this SOAP request and passes it off to the class for processing. The internals of .Net take care of instantiating the Web Service class, firing the constructor and then invoking the requested method, passing in the provided parameters, which are properly converted into the types specified by the WSDL file. This entire process is taken care of by the .Net Framework and is not exposed to the developer.

At this point the method executes. This is where all the user processing occurs. We have written a custom method code to implement business logic required to service the functionality of the Web Service. Once this is done, the result value is returned as a simple return from the method.

The .Net Web Service handlers then take the return value and package it up into a SOAP response packet and send this output back over the HTTP connection to the client application proxy. The proxy now picks up the SOAP response, and unpacks the SOAP response, performs the type conversion of the XML back into the proper return type specified in the WSDL document and returns the value back to the client side. This action happens on the fly, and is taken care of by the .Net infrastructure. .Net built in classes take care of XML serialization/deserialization. System.Xml.Serialization.Serializer class takes care of XML serialization. This class has a method called serializer that does this procedure. A method called deserializer in this class does the deserialization. Fig 3.1 below shows the web services interactions. On the client side, we have separate classes for different predictor methods: ShortTermPage.cs and LongTermPage.cs. The former invokes the GetShortTermPrediction method whereas the latter invokes the GetLongTermPrediction method both of which are provided by the web service. Research.cs is a class that invokes GetHistory method provided by the web service to show the stock history results to the user. Similarly, from the web service end, the reverse action takes place. To keep it compact, our interaction diagram shows only the movement of one of these methods, however, the same procedure is followed by each of the methods. “Method Calls” refer to each of the methods being called from the client end. “Web methods” denote the methods defined by the web service.

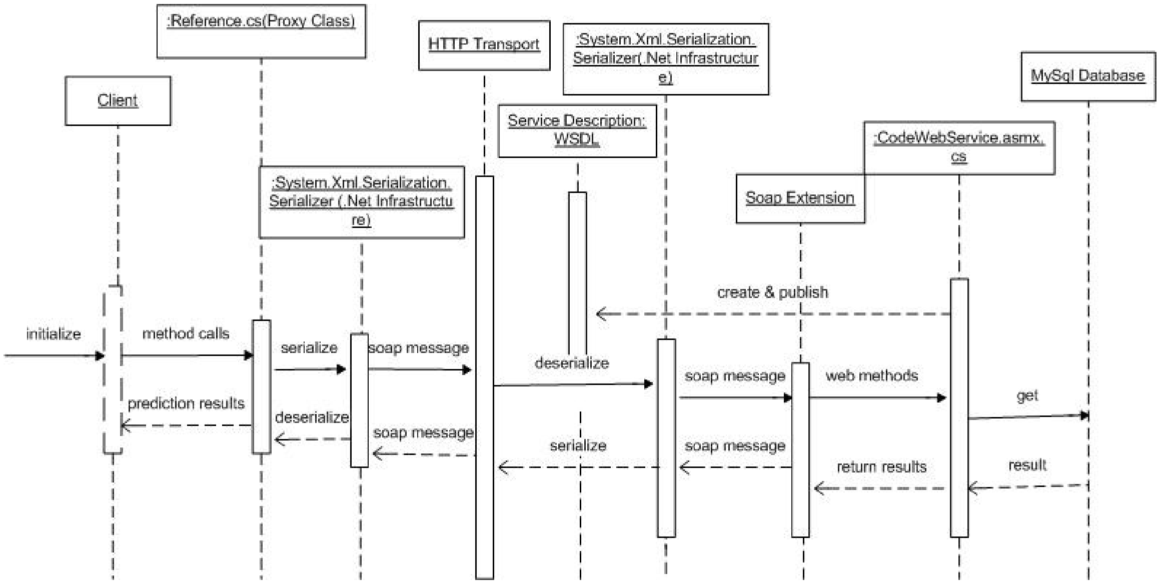


Fig 3.1 Web Services Interactions.

In our Web Service, we have included four main web methods. The first method GetShortTermPrediction returns the short term prediction values namely, predicted price, value at risk and the prediction whether to buy, sell or hold for a particular company. This method takes an argument of type TickerInfo and returns these decision parameters from the database for the corresponding ticker. TickerInfo is a class that holds the properties, namely, id and name of a ticker. The second method GetLongTermPrediction returns the recognized pattern based on historical data and the predicted decision whether to buy, sell or hold from the database, for a particular symbol requested by the client. GetAllTickers method returns all the tickers from the database. GetHistory method returns stock history data from the database. Figure 3.2 below shows the different classes in the WebServices module.

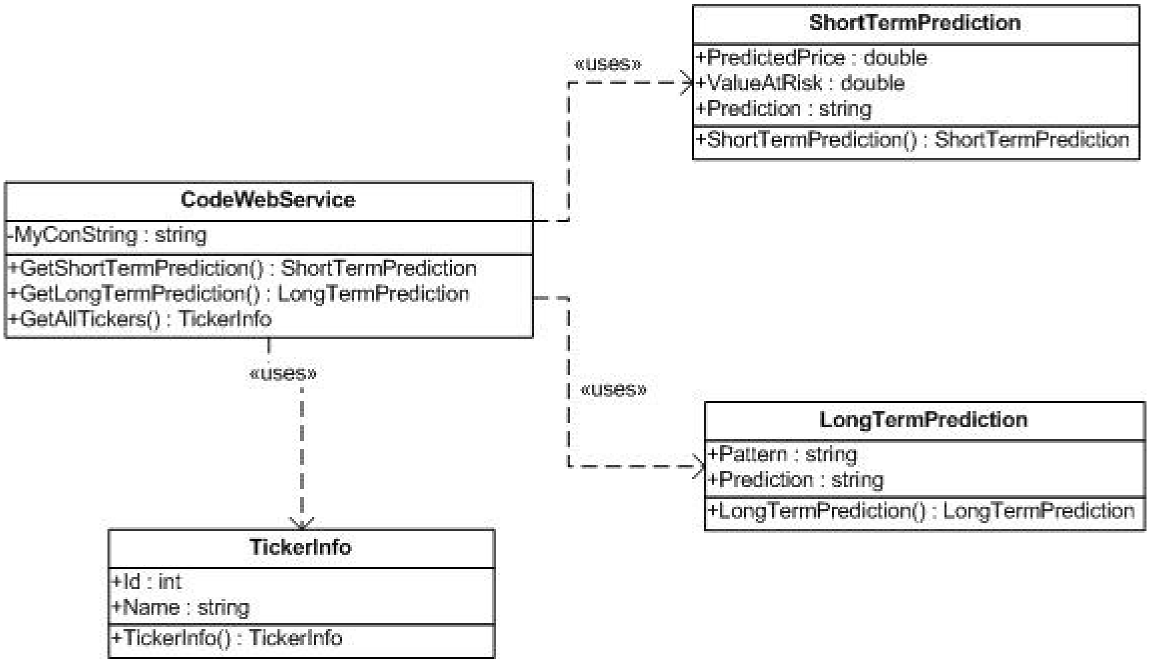
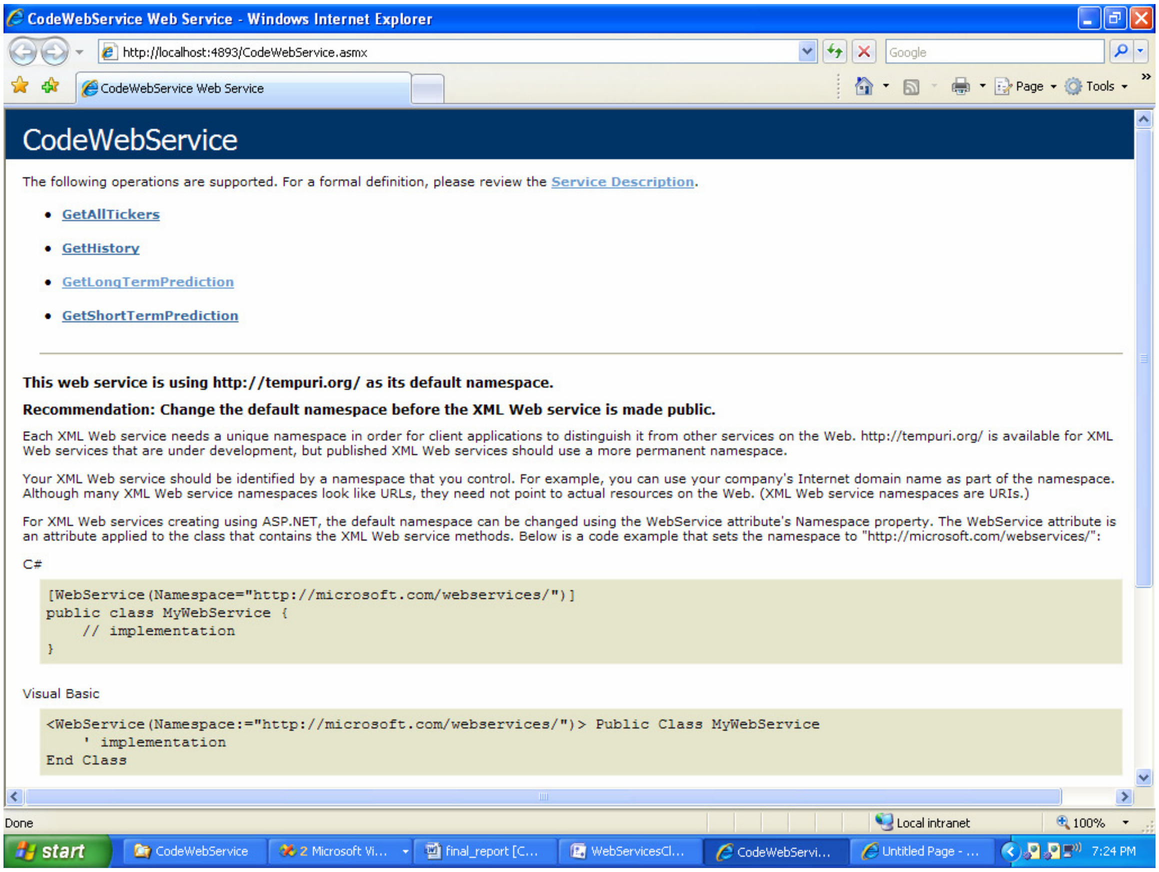

ShortTermPrediction
+PredictedPrice : double
+ValueAtRisk : double
+Prediction : string
+ShortTermPrediction() : ShortTermPrediction
«uses»
CodeWebService
-MyConString : string
+GetShortTermPrediction() : ShortTermPrediction
+GetLongTermPrediction() : LongTermPrediction
+GetAllTickers() : TickerInfo
«uses»
«uses»
LongTermPrediction
+Pattern : string
+Prediction : string
+LongTermPrediction() : LongTermPrediction
TickerInfo
+Id : int
+Name : string
+TickerInfo() : TickerInfo

Fig 3.2 : Web Services Class Diagram

Fig 3.3 shows the screenshot when the web service is invoked. This screen displays all methods defined in the web service. On clicking “Service Description” in this page (right top corner), we see the WSDL file description. Fig 3.4 shows the WSDL file. SOAP messages for GetLongTermPrediction is shown in Fig 3.5. SOAP messages for GetShortTermPrediction is shown in Fig 3.6.



CodeWebService

The following operations are supported. For a formal definition, please review the Service Description.

- GetAllTickers
- GetHistory
- GetLongTermPrediction
- GetShortTermPrediction

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Visual Basic

```
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
    ' implementation
End Class
```

Fig 3.3 Screenshot of web service interface

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://tempuri.org/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://tempuri.org/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
      <s:element name="GetShortTermPrediction">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="ticker" type="tns:TickerInfo" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="TickerInfo">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Id"
type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="Name"
type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
name="Symbol" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetShortTermPredictionResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetShortTermPredictionResult"
type="tns:ShortTermPrediction" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ShortTermPrediction">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1"
name="PredictedPrice" type="s:double" />
          <s:element minOccurs="1" maxOccurs="1"
name="ValueAtRisk" type="s:double" />
          <s:element minOccurs="0" maxOccurs="1"
name="Prediction" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetLongTermPrediction">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="ticker" type="tns:TickerInfo" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetLongTermPredictionResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetLongTermPredictionResult"
type="tns:LongTermPrediction" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="LongTermPrediction">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1"
name="Pattern" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1"
name="Prediction" type="s:string" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetAllTickers">
        <s:complexType />
      </s:element>
      <s:element name="GetAllTickersResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetAllTickersResult" type="tns:ArrayOfTickerInfo" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfTickerInfo">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded"
name="TickerInfo" nillable="true" type="tns:TickerInfo" />
        </s:sequence>
      </s:complexType>
      <s:element name="GetHistory">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="ticker" type="tns:TickerInfo" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetHistoryResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="GetHistoryResult" type="tns:ArrayOfTickerHistory" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfTickerHistory">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded"
name="TickerHistory" nillable="true"
type="tns:TickerHistory" />
        </s:sequence>
      </s:complexType>
      <s:complexType name="TickerHistory">
        <s:sequence>
          <s:element minOccurs="1" maxOccurs="1" name="Dt"
type="s:dateTime" />
          <s:element minOccurs="1" maxOccurs="1" name="Open"
type="s:decimal" />
          <s:element minOccurs="1" maxOccurs="1" name="High"
type="s:decimal" />
          <s:element minOccurs="1" maxOccurs="1" name="Low"
type="s:decimal" />
          <s:element minOccurs="1" maxOccurs="1"
name="Close" type="s:decimal" />
```

```
          <s:element minOccurs="1" maxOccurs="1"
name="Price" type="s:decimal" />
          <s:element minOccurs="1" maxOccurs="1"
name="Volume" type="s:long" />
        </s:sequence>
      </s:complexType>
    </s:schema>
  </wsdl:types>
  <wsdl:message name="GetShortTermPredictionSoapIn">
    <wsdl:part name="parameters"
element="tns:GetShortTermPrediction" />
  </wsdl:message>
  <wsdl:message name="GetShortTermPredictionSoapOut">
    <wsdl:part name="parameters"
element="tns:GetShortTermPredictionResponse" />
  </wsdl:message>
  <wsdl:message name="GetLongTermPredictionSoapIn">
    <wsdl:part name="parameters"
element="tns:GetLongTermPrediction" />
  </wsdl:message>
  <wsdl:message name="GetLongTermPredictionSoapOut">
    <wsdl:part name="parameters"
element="tns:GetLongTermPredictionResponse" />
  </wsdl:message>
  <wsdl:message name="GetAllTickersSoapIn">
    <wsdl:part name="parameters" element="tns:GetAllTickers"
/>
  </wsdl:message>
  <wsdl:message name="GetAllTickersSoapOut">
    <wsdl:part name="parameters"
element="tns:GetAllTickersResponse" />
  </wsdl:message>
  <wsdl:message name="GetHistorySoapIn">
    <wsdl:part name="parameters" element="tns:GetHistory" />
  </wsdl:message>
  <wsdl:message name="GetHistorySoapOut">
    <wsdl:part name="parameters"
element="tns:GetHistoryResponse" />
  </wsdl:message>
  <wsdl:portType name="CodeWebServiceSoap">
    <wsdl:operation name="GetShortTermPrediction">
      <wsdl:input message="tns:GetShortTermPredictionSoapIn"
/>
      <wsdl:output
message="tns:GetShortTermPredictionSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetLongTermPrediction">
      <wsdl:input message="tns:GetLongTermPredictionSoapIn"
/>
      <wsdl:output
message="tns:GetLongTermPredictionSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetAllTickers">
      <wsdl:input message="tns:GetAllTickersSoapIn" />
      <wsdl:output message="tns:GetAllTickersSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetHistory">
      <wsdl:input message="tns:GetHistorySoapIn" />
      <wsdl:output message="tns:GetHistorySoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="CodeWebServiceSoap"
type="tns:CodeWebServiceSoap">
    <soap:binding
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetShortTermPrediction">
      <soap:operation
soapAction="http://tempuri.org/GetShortTermPrediction"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetLongTermPrediction">
      <soap:operation
soapAction="http://tempuri.org/GetLongTermPrediction"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetAllTickers">
      <soap:operation
soapAction="http://tempuri.org/GetAllTickers"
style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetHistory">
      <soap:operation
soapAction="http://tempuri.org/GetHistory" style="document"
/>
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:binding name="CodeWebServiceSoap12"
type="tns:CodeWebServiceSoap">
    <soap12:binding
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="GetShortTermPrediction">
      <soap12:operation
soapAction="http://tempuri.org/GetShortTermPrediction"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetLongTermPrediction">
      <soap12:operation
soapAction="http://tempuri.org/GetLongTermPrediction"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetAllTickers">
      <soap12:operation
soapAction="http://tempuri.org/GetAllTickers"
style="document" />
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="GetHistory">
      <soap12:operation
soapAction="http://tempuri.org/GetHistory" style="document"
/>
      <wsdl:input>
        <soap12:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap12:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CodeWebService">
    <wsdl:port name="CodeWebServiceSoap"
binding="tns:CodeWebServiceSoap">
```

```
      <soap:address
location="http://localhost:4893/CodeWebService.asmx" />
    </wsdl:port>
    <wsdl:port name="CodeWebServiceSoap12"
binding="tns:CodeWebServiceSoap12">
      <soap12:address
location="http://localhost:4893/CodeWebService.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Fig 3.4 WSDL

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CodeWebService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetLongTermPrediction"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetLongTermPrediction xmlns="http://tempuri.org/">
      <ticker>
        <Id>int</Id>
        <Name>string</Name>
        <Symbol>string</Symbol>
      </ticker>
    </GetLongTermPrediction>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
```

```
    <GetLongTermPredictionResponse
xmlns="http://tempuri.org/">
      <GetLongTermPredictionResult>
        <Pattern>string</Pattern>
        <Prediction>string</Prediction>
      </GetLongTermPredictionResult>
    </GetLongTermPredictionResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP 1.2

The following is a sample SOAP 1.2 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CodeWebService.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetLongTermPrediction xmlns="http://tempuri.org/">
      <ticker>
        <Id>int</Id>
        <Name>string</Name>
        <Symbol>string</Symbol>
      </ticker>
    </GetLongTermPrediction>
  </soap12:Body>
</soap12:Envelope>
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetLongTermPredictionResponse
xmlns="http://tempuri.org/">
```

```
      <GetLongTermPredictionResult>
        <Pattern>string</Pattern>
        <Prediction>string</Prediction>
      </GetLongTermPredictionResult>
    </GetLongTermPredictionResponse>
  </soap12:Body>
</soap12:Envelope>
```

Fig 3.5 SOAP 1.1 and 1.2 for GetLongTermPrediction

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CodeWebService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetShortTermPrediction"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetShortTermPrediction xmlns="http://tempuri.org/">
      <ticker>
        <Id>int</Id>
        <Name>string</Name>
        <Symbol>string</Symbol>
      </ticker>
    </GetShortTermPrediction>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetShortTermPredictionResponse xmlns="http://tempuri.org/">
      <GetShortTermPredictionResult>
        <PredictedPrice>double</PredictedPrice>
        <ValueAtRisk>double</ValueAtRisk>
        <Prediction>string</Prediction>
      </GetShortTermPredictionResult>
    </GetShortTermPredictionResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP 1.2

The following is a sample SOAP 1.2 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /CodeWebService.asmx HTTP/1.1
Host: localhost
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetShortTermPrediction xmlns="http://tempuri.org/">
      <ticker>
        <Id>int</Id>
        <Name>string</Name>
        <Symbol>string</Symbol>
      </ticker>
    </GetShortTermPrediction>
  </soap12:Body>
</soap12:Envelope>
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: length

<?xml version="1.0" encoding="utf-8"?>
<soap12:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
  <soap12:Body>
    <GetShortTermPredictionResponse xmlns="http://tempuri.org/">
      <GetShortTermPredictionResult>
        <PredictedPrice>double</PredictedPrice>
        <ValueAtRisk>double</ValueAtRisk>
        <Prediction>string</Prediction>
      </GetShortTermPredictionResult>
    </GetShortTermPredictionResponse>
  </soap12:Body>
</soap12:Envelope>
```

Fig 3.6 SOAP 1.1 and 1.2 for GetShortTermPrediction

4. Stock Prediction using Machine Learning Techniques

Stock market prediction has been an important issue in the field of finance and mathematics due to its potential financial gain. As a vast amount of capital is traded through the stock market, the stock market is seen as a peak investment outlet. In addition, stock market prediction brings with it the challenge of proving whether the financial market is predictable or not. With the advent of faster computers and vast information over the Internet, stock markets have become more accessible to either strategic investors or the general public.

With advent of Machine Learning techniques, computers can “learn” various features and trends in data which can then be used to predict future values. The major focus of machine learning research is to extract information from data automatically, by computational and statistical methods. Various machine learning techniques like Support Vector Machines (SVMs) and Neural Networks (NNs) have been studied to analyze their capability to predict the stock markets.

An important requirement to be considered while designing the Stock predictor is the investment horizon of the user i.e. whether the user wishes for short term or long term investment. This issue is critical as one algorithm may suite short term investment strategy while it may fail miserably when applied for long term gains and vice versa. Keeping this in mind we have implemented two modules viz. SVM based long term predictor which helps the user for long term gains in a volatile market and Monte Carlo Predictor for short term gains in the market. For long term prediction a large amount of data needs to be analyzed. SVMs are more suited for long term prediction because they are easy to train, computationally less intensive and efficient [4][5][6] as compared to other techniques like Neural Networks. The comparison of SVMs and Neural Network follows in next few sections. Also Monte Carlo method uses the correlation in stocks in a particular sector to predict the future values. As correlation in stock data holds good for short time and may or may not hold good for long periods, we use Monte Carlo method for short term prediction. Before doing so we present a sort of literature study carried out before implementing the algorithms.

4.1 Support Vector Machine based long term Stock Predictor:

There are various machine learning algorithms which are used for Stock prediction but amongst them the two most successful and commonly used are Support Vector Machines (SVMs) and Neural Networks (NNs). We carried out an initial literature survey of the two algorithms to figure out the best algorithm which suites our requirements.

Support Vector Machines:

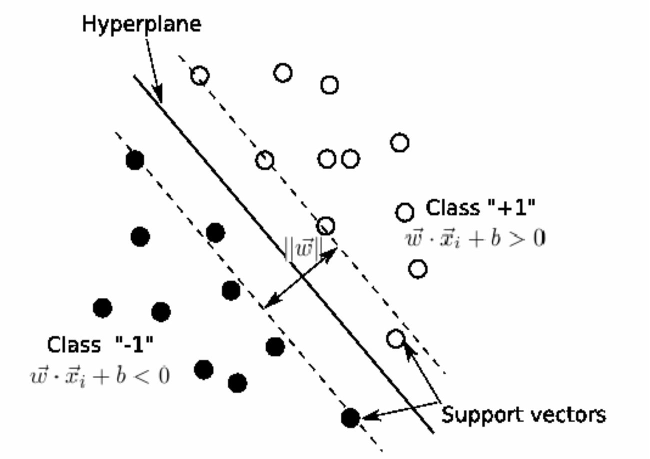

Hyperplane
Class "+1"
$\vec{w} \cdot \vec{x}_i + b > 0$
$||\vec{w}||$
Class "-1"
$\vec{w} \cdot \vec{x}_i + b < 0$
Support vectors

Fig. 4.1:Support Vector Machine classifier.

A Classifier is a function that maps input data samples to a defined set of object class. The task of the Classifier is to determine the value of the function for any valid input object after having seen a number of training examples (i.e., pairs of input and target output).

The classifier has to assign the input measurements to one of the classes learned in the training phase. In SVMs [1][2], this classification of data is achieved by defining a plane known as a “hyperplane” that separates the data samples of different class (Fig.4.1). The closest data points to the hyperplane are called Support Vectors (SVs). The hyperplane is determined in a way to maximize the distance from the nearest data point to the plane. The maximized distance introduced, accommodates for any minor deviations
in the data and also increases the ability of the classifier to learn. The hyperplane is of the form, w. x - b= 0, where w is the distance between the plane and the closest data point, x is the input data and b is the offset from the origin.

4.2 Neural Networks:

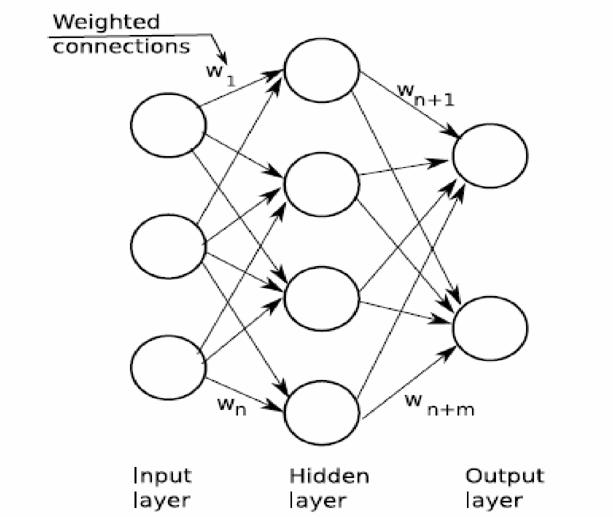


Fig 4.2 Neural Networks classifier.

A NN [1][3] is a computational model based on biological neural networks. NN consists of interconnecting artificial neurons or nodes (programming constructs that mimics the properties of biological neurons) to classify the input data into defined object class. Generally, NNs have three layers; input, output and hidden layer (Fig.4.2). Each layer receives one or more inputs (object characteristics) and sums them to produce an output value that represents a defined object class. Usually the sums at each node are weighted based on the connecting branch.

A NN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. During learning phase, the sample records are processed and the output is compared with the expected classification of the record. The resulting error in the decision is used as a feedback to adjust weights associated to nodes. This is done iteratively until the system successfully classifies all the sample data. The utility of NN models lies in the fact that it can be used to infer a function (that classifies input data) based on training samples.

4.3 Support Vector Machines Vs Neural Networks:

Comparative studies to analyze the capabilities of the two classifiers are presented by researchers [4][5][6][7]. NNs use feedback during training phase which may lead to errors due to iterative training while SVMs use mathematical kernels to separate the data, providing a global and unique solution. NN requires a skillful human to validate the

results of classification during the training phase. NNs have the black box problem, which does not reveal the significance of each variable and the way they weigh independent variables. As the individual role of each variable cannot be determined, it is impossible to understand
how the network produces future price of stock.

The main disadvantage of NN is that the weights determined to do classification tends to work for input data similar to training samples used. That is, NN tends to overfit the data set, reducing the learning and generalization capability of the network. In SVMs, the problem of overfitting doesn't occur because a mathematical kernel is used. Thus, SVMs are more suitable for Stock prediction than NNs.

4.4 Conclusions:

As concluded from the above comparison, Support Vector Machines are more suited to the problem of Stock Prediction as compared to Neural Networks. We use a backend Predictor engine in matlab which will carry out the long term predictions using Support Vector Machines. For SVMs we use a Machine learning tool box called "SPIDER" [8] which will help in training of SVMs.

4.5 Package Diagram:

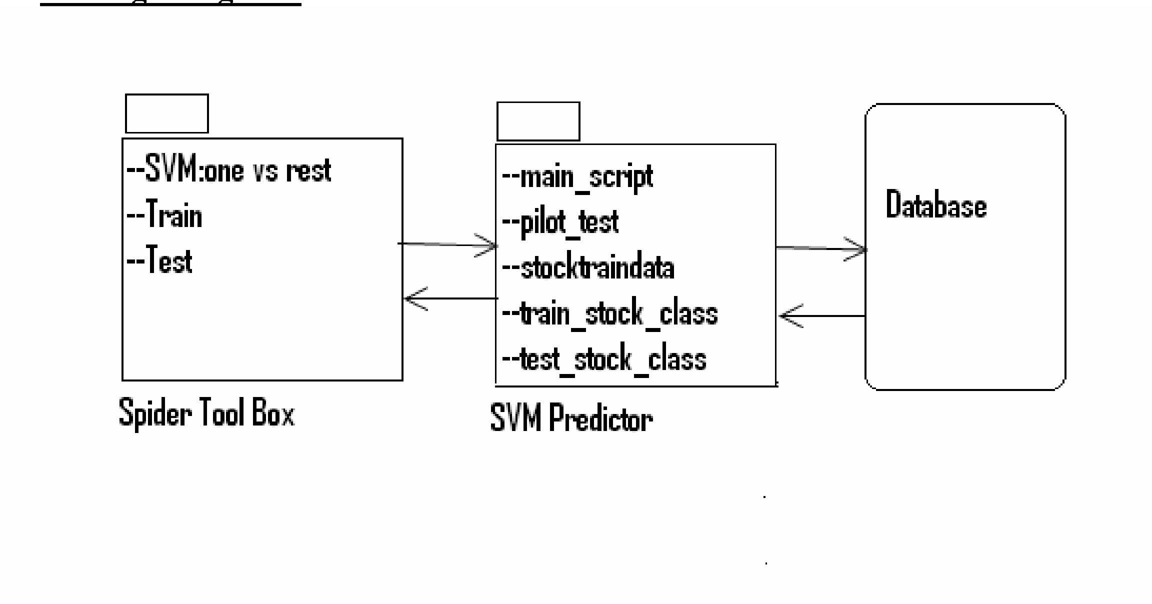


Fig 4.3 Package Diagram For the SVM Based Long Term Predictor.

As shown in the package diagram we have three packages; SPIDER toolbox package, SVM Pattern predictor package and the database.

SPIDER Machine Learning Toolbox:

SPIDER is a matlab toolbox for machine learning algorithms. It has various machine learning algorithms and variations in SVMs. We will be training our Support Vector Machine using the **train** and **test** functionality provided by SPIDER. As we plan to predict 4 patterns in the data we use **one_vs_rest algorithm** [1][2] to find out the Support Vectors and the hyper plane as discussed in earlier sections.

SVM based Pattern Predictor:

SVM pattern predictor package has 5 functions as follows,

Stocktraindata: This function provides sample data patterns to train our SVM. It has data resembling to Cup and Saucer, Ascending Triangle, Descending Triangle and Head and Shoulders [15][16][19]. Extending the number of patterns to be predicted can very easily be achieved by including the data patterns in this function. Thus this function provides easy scaling and functionality of the application.

train_stock_class: This function interacts with the SPIDER toolbox and provides the necessary parameters to train the SVM. We use one_vs_rest algorithm, Radial Basis Kernel and cross validation [1][2] to make our SVM decision robust. The parameter details are very specific to Support Vector Machines and our out of scope of this document. Users are recommended to go through [1][2] which gives a good insight into the parameters used. For our application we use optimal parameter which suite our goal of Pattern prediction.

test_stock_class: This function interacts with the SPIDER toolbox and provides it with the trained SVM model and the test data i.e. the data whose pattern is to be predicted. In turn this function receives the pattern detected in the data.

pilot_test: This function aggregates the information received from test_stock_class into a decision for “BUY”, “SELL” or “HOLD” the stock. Thus relating the pattern detected with the recommended action to be taken by the User.

Main_script: This is the main function which takes care of interaction with the database and the order of the execution of various functions.

Database: The table historical and lt_val are used by the SVM predictor to get the previous stock values and update its decision.

4.6 System Design and Prediction Algorithm

Design Considerations and Assumptions:

1. **Matlab:** We use matlab as a backend engine for SVM based long term predictor. SPIDER machine learning toolbox is widely used for implementing machine learning algorithms. Although matlab engine requires more resources, it is

invoked periodically once a day. Thus, reducing the overall operating cost of the system. This is based on a valid assumption that pattern in data changes over several months.

2. **Train data:** One important concern is the quality of data used to train the SVM. Due to lack of quality data available for free on the internet, we simulated four data patterns viz. Cup and Saucer, Ascending Triangle, Descending Triangle and Head and Shoulders synthetically as shown in the fig. The SVM is trained on numerous 12-valued synthetically generated pattern data. The 12-element vector represents the monthly average of stock values over a year. An important assumption here is that pattern can be recognized if it exists over a time frame of 1 year.

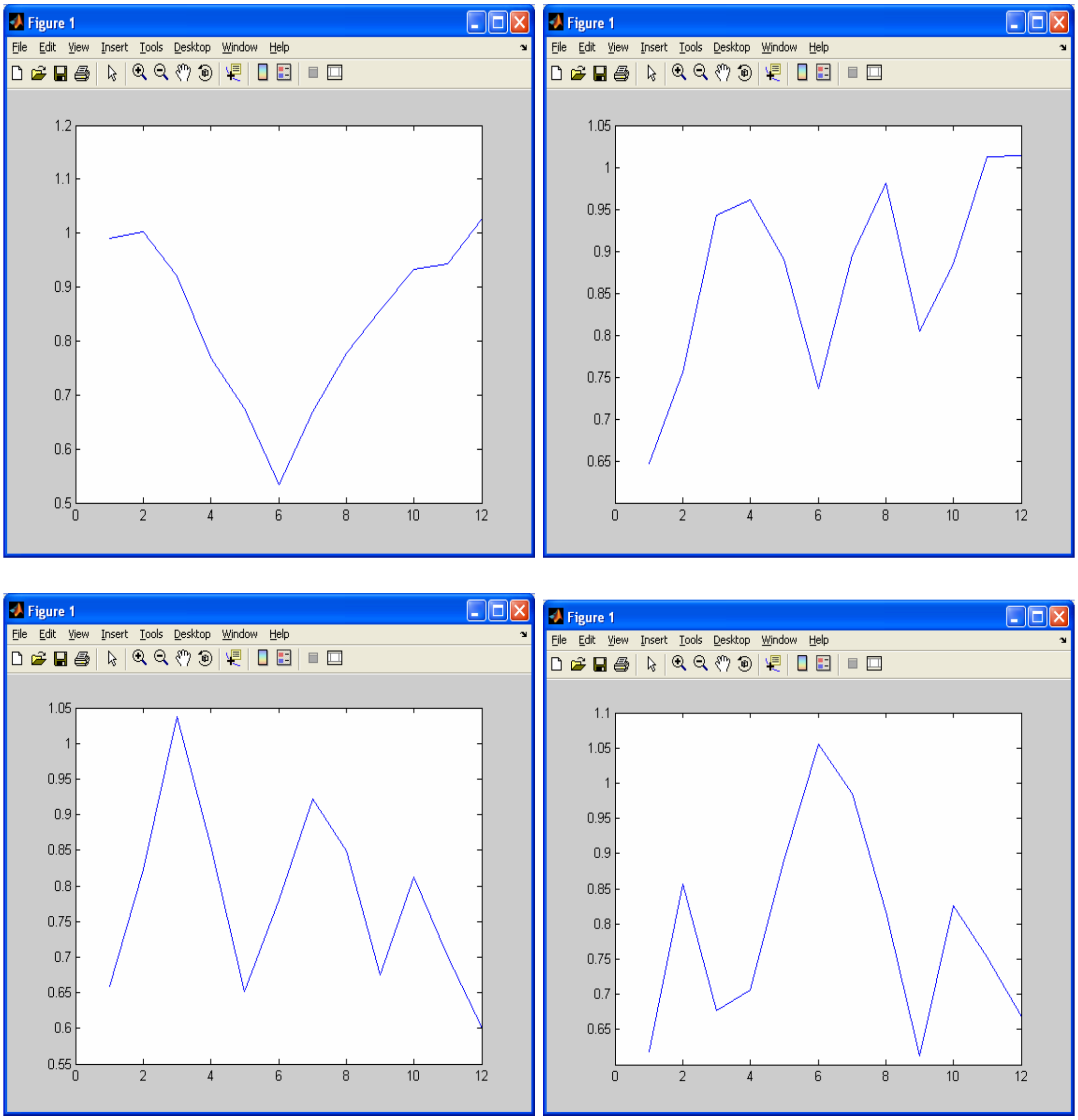


Fig: Illustrate Cup and Saucer, Ascending Triangle, Descending Triangle and Head and Shoulder used to train the SVM.

3. **Patterns Predicted:** System is trained to predict Cup and Saucer, Ascending Triangle, Descending Triangle and Head and Shoulders patterns in data[15][16][19]. These basic patterns are studied by data analysts to identify a trend in stock data and predict the future price of the stocks. We have selected these patterns because some of the patterns are positive indicators which indicate “BUY” while some of them are negative indicators which indicate “SELL”. The number of patterns to be recognized can easily be increased by feeding the sample data to stocktraindata file.

Algorithm:

4. The matlab SVM predictor engine is invoked periodically once daily to predict patterns. This is based on a valid assumption that pattern in data changes over several months.
5. The main matlab script fetches 1 year historical stock data for 48 companies from the database.
6. The data is pre-processed to compute average over each month which leads to a 12 valued pre processed and normalized vector.
7. The SVM is trained on the different data patterns which already resides (stocktraindata) on the matlab engine returning the learnt model i.e. learnt Support Vectors and hyperplane.
8. Preprocessed data along with the learnt parameters are then passed to the SVM to detect patterns and come up with a decision.
9. The predicted pattern and the corresponding recommended action “BUY”, “SELL” or “HOLD” is updated in the database.

Flowchart:

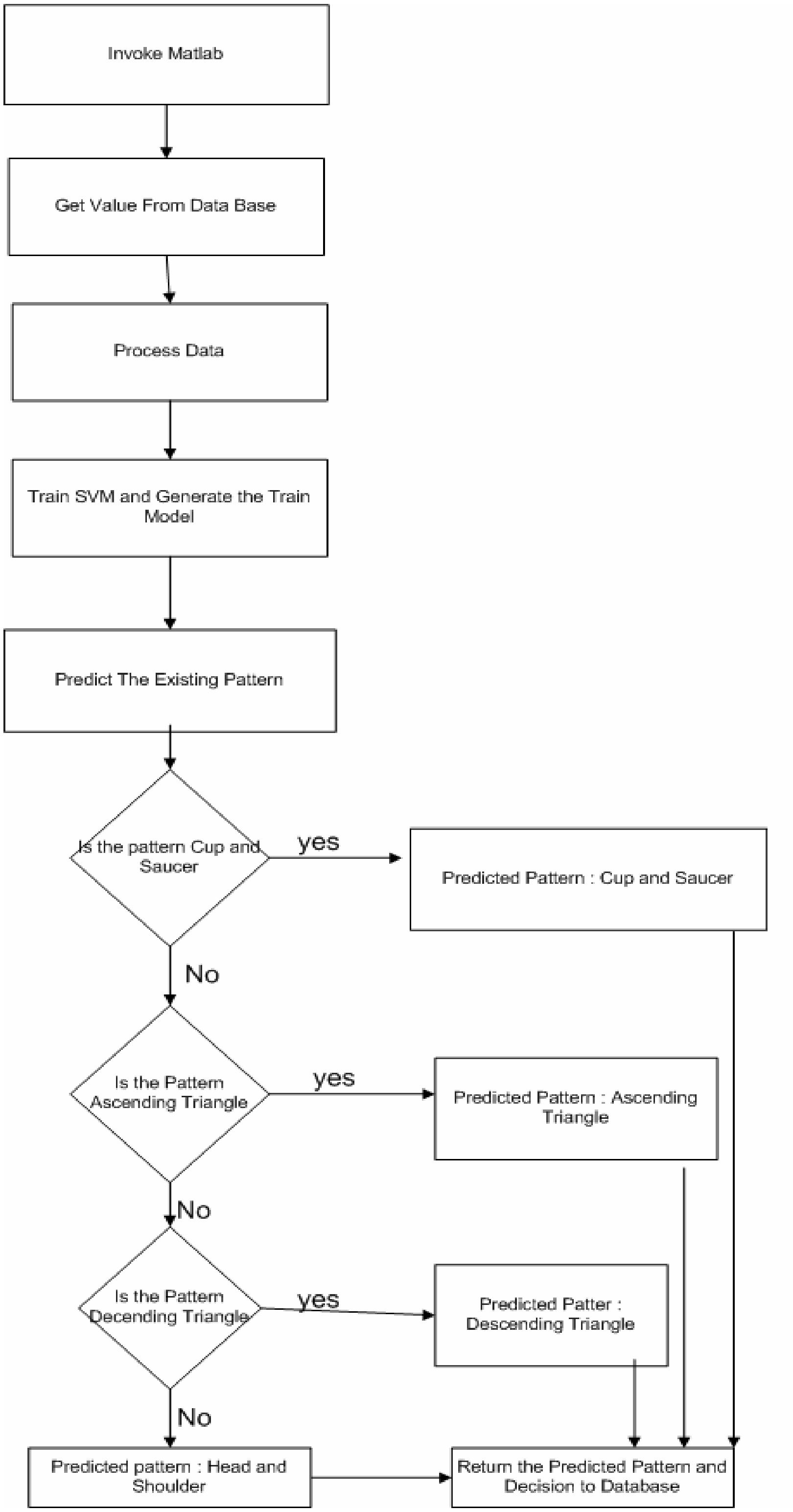

Invoke Matlab
Get Value From Data Base
Process Data
Train SVM and Generate the Train Model
Predict The Existing Pattern
Is the pattern Cup and Saucer
yes
Predicted Pattern : Cup and Saucer
No
Is the Pattern Ascending Triangle
yes
Predicted Pattern : Ascending Triangle
No
Is the Pattern Decending Triangle
yes
Predicted Patter : Descending Triangle
No
Predicted pattern : Head and Shoulder
Return the Predicted Pattern and Decision to Database

Fig 4.4 SVM Algorithm.

4.7 SVM Predictor Features:

1. Support Vector Machine pattern detector provides more insight into analyzed data and is used to predict pattern and decision from long term investment perspective.
2. SVM used for prediction task is more superior and robust than its existing peers like Neural Networks increasing the confidence level in the recommended decisions.
3. Pattern detection in stock data is very difficult and computationally intensive task [20][21] as they vary over a long period and large amount of data needs to be processed. But the use of SVM reduces the computational costs tremendously.
4. Although matlab engine requires more resources, it is invoked once a day. Thus, reducing the overall operating cost of the system.

5. Monte Carlo Predictor

5.1 Value-at-Risk

Value-at-Risk is a very general concept that has broad applications, it is most commonly used by security firms or investment banks to measure the market risk of their asset portfolios (market value at risk) [23]. VAR is widely applied in finance for quantitative risk management for many types of risk. In economics and finance, Value at Risk (VaR) is the maximum loss not exceeded with a given probability defined as the confidence level, over a given period of time.

The most popular and traditional measure of risk is volatility [13]. The main problem with volatility, however, is that it does not care about the direction of an investment's movement: a stock can be volatile because it suddenly jumps higher. For investors, risk is about the odds of losing money, and VAR is based on that common-sense fact. By assuming investors care about the odds of a really big loss, VAR answers the question, "What is my worst-case scenario?" or "How much could I lose in a really bad month?"

A VAR statistic has three components: a time period, a confidence level and a loss amount (or loss percentage). Keeping these three parts in mind; the following are some examples of variations of the question that VAR answers:

- What is the most I can - with a 95% or 99% level of confidence - expect to lose in dollars over the next month?
- What is the maximum percentage I can - with 95% or 99% confidence - expect to lose over the next year?

There are three methods of calculating VAR namely the historical method, the variance-covariance method and the Monte Carlo simulation.

5.2 Historical Method

The historical method simply re-organizes actual historical returns, putting them in order from worst to best [13]. It then assumes that history will repeat itself, from a risk perspective.

As an example we will consider the Nasdaq 100 Index, which trades under the ticker QQQ. We calculate each daily return of QQQ (QQQ started trading in Mar 1999) and produce a rich data set of almost 1,400 points shown in figure 5.1.

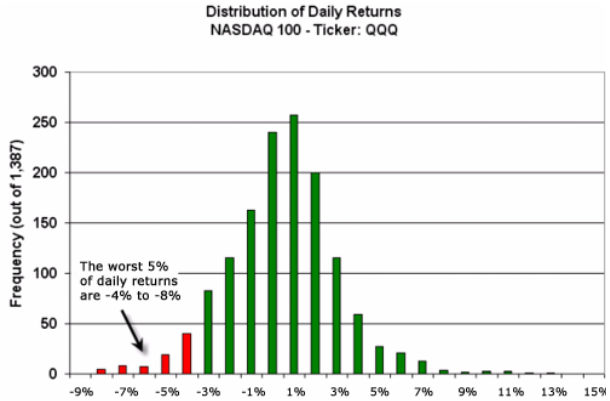


Fig 5.1 The red bars that compose the "left tail" correspond to the lowest 5% daily returns. Hence, with 95% confidence we can say that the worst daily loss will not exceed 4% (Courtesy: Investopedia).

The red bars that compose the "left tail" of the histogram are the lowest 5% of daily returns (since the returns are ordered from left to right, the worst is always the "left tail"). The red bars run from daily losses of 4% to 8%. Because these are the worst 5% of all daily returns, we can say with 95% confidence that the worst daily loss will not exceed 4%. Put another way, we expect with 95% confidence that our gain will exceed -4%.

5.3 The Variance-Covariance Method

The Variance-Covariance method assumes that stock returns are normally distributed [13]. In other words, it requires that we estimate only two factors - an expected (or average) return and a standard deviation - which allow us to plot a normal distribution curve.

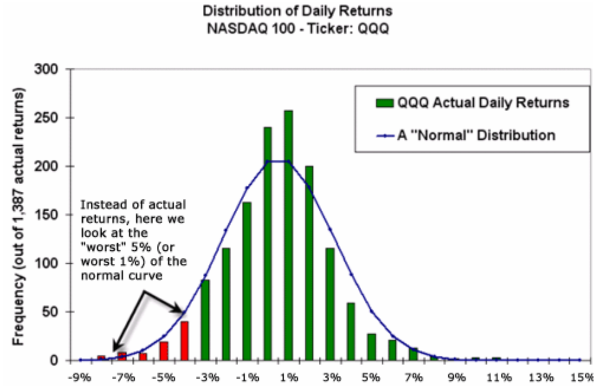
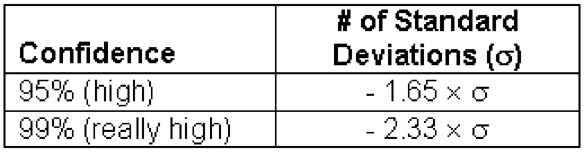


Fig 5.2 A normal distribution of the daily returns is depicted by the blue line. The lowest 5% of daily returns is calculated based on the normal distribution curve (Courtesy: Investopedia).

The idea behind the variance-covariance is similar to the ideas behind the historical method - except that we use the familiar curve instead of actual data. The advantage of the normal curve is that we automatically know where the worst 5% and 1% lie on the curve. They are a function of our desired confidence and the standard deviation (σ):



Confidence	# of Standard Deviations (σ)
95% (high)	- 1.65 × σ
99% (really high)	- 2.33 × σ

5.4 Monte Carlo Simulation

The third method which is the method I have employed in calculating VAR involves developing a model for future stock price returns and running multiple hypothetical trials through the model. A Monte Carlo simulation refers to any method that randomly generates trials, but by itself does not tell us anything about the underlying methodology. The model considered here consists of an over simplified *Black-Scholes equation* [9].

The goal is to simulate prices of stocks at time *t+1* given stock prices up to time *t*. We represent this distribution as a set of 1000 possible values called "Simulated Prices". The input to this calculation is historical prices of the stocks. Based on the trailing year's worth of prices, we calculate the variance and covariance of returns. Based on variance and correlation of returns, we then calculate 1000 equally probable sets of prices for this group of stocks one day forward. Each of these sets of prices is one scenario in our Monte-Carlo simulation.

Below are the steps to predict the Value at Risk using Monte Carlo Simulations [9]:

Step 1: Historical Returns

Given a portfolio of stocks we calculate the historical returns from the historical prices. In risk we typically deal in historical returns rather than historical prices. The primary reason is that the math is much easier if we assume that processes are stationary (moments are constant over time). While this assumption is not always true, the effects of adopting this assumption are minimal in this context. If we represent the price today as p_t, the price yesterday as $p_{(t-1)}$, and the return between yesterday and today as r_t. The returns are calculated using:

$$r_t = \left[{(p_t - p_{(t-1)})}\big/{p_{(t-1)}} \right] \times 100$$

Step 2: Volatility

In financial context, volatility is synonymous with standard deviation. We calculate the volatility of each stock price using

$$\sigma = \sqrt{E\left(\left(X - E(X)\right)^2\right)}$$

Where, X is a random variable consisting of the stocks and $E(X)$ is the expected value of X (another word for mean), $E(X) = \frac{\sum_{i=1}^{n} x_i}{n}$

Step 3: Variance-Covariance Matrix

We create a matrix that represents the variance and covariance of the stock returns. Over short periods of time, the volatility of returns dominates the mean of returns. As such, we make the simplifying assumptions that, over the one day period of interest, the mean stock return is zero [9]. With mean of 0, the variance is defined as

$$\sigma_X^2 = E(X^2)$$

And the Covariance is defined as

$$\sigma_{XY} = E(XY)$$

In matrix form, Variance-Covariance can be calculated as

$$\sum = X'X/T$$

Where $\sum$ is the commonly used notation for the Variance-Covariance matrix, X is the $T \times n$ matrix of security returns, n is the number of securities, and T is the number of historical returns.

Step 4: IID Normal Random Numbers, Cholesky Decomposition, Simulated Returns

We use the Variance-Covariance matrix to calculate 1000 equally likely random sets of returns for our n securities. To do this we generate 1000 sets of IID Normal Random Numbers each with a mean of 0 and standard deviation of 1 $(N(0,1))$. To transform a set Z with $N(0,1)$ random variables to a set of random variables with mean μ and variance σ^2, we can apply the following transform:

$$Z' = \sigma(Z + \mu)$$

The resultant is a set of IID Normal Random Numbers Z' with mean μ and variance σ^2 $(N(\mu, \sigma^2))$.

To do this with a set of normally distributed, linearly correlated random variables, the following is applied in matrix form:

$$X' = W(X + diag(\mu_i))$$

Since we are assuming that $\mu = 0$, this reduces to

$$X' = WX$$

Where X is a $m \times n$ matrix of IID $N(0,1)$random numbers, n is the number of securities, and m is the number of Monte-Carlo scenarios chosen. X' is an $m \times n$ matrix of simulated return of $N(0, \sigma^2)$.

$W$is the Cholesky Decomposition of $\sum$ which is calculated as follows:

$$\sum = LDL^T$$

Where $\sum$ is the symmetric $n \times n$ Variance-Covariance Matrix, which is decomposed in to a $n \times n$ diagonal Matrix D, an $n \times n$ lower triangular matrix L and an $n \times n$ upper triangular matrix L^T

The Cholesky decomposition matrix is calculated as:

$$W = \sqrt{D}L^T$$

We use this method to calculate the Simulated Returns from the IID Normal Random Numbers $N(0,1)$, and the Variance-Covariance Matrix $\sum$.

Step 5: Simulated Prices

The simulated prices S are calculated from the simulated returns X'. The simulated price varies exponentially and is calculated using:

$$s_i = p_t \times e^{x_i}$$

Where, p_t is the current stock price and x_i are the m simulated returns.

Step 6: Option Pricing

Each of the options represents a single European style call equity stock option. We use the Black-Scholes-Merton [14] equation to price call options on a stock. We calculate the current value of the stock option. We then calculate the value of the stock option as if it were aged to tomorrow and the stock price changed as we calculated in the previous step. Since we calculated the value tomorrows stock price 1000 times, we calculate the value of the option 1001 times (base case + 1000 Monte-Carlo scenarios). Each of the changes (represented by the Unit Change column) is multiplied by the position size to get a Profit & Loss (P&L) number for the position under each of the 1000 scenarios.

The following Black-Scholes-Merton formula for the call price C of a European call option is used to simulate the call price of each of the simulated stock prices:

$$C(S,T) = S \times \Phi(d_1) - K \times e^{-rT}\Phi(d_2)$$

Where,

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2) \times T}{\sigma \times \sqrt{T}}$$

$$d_2 = d_1 - \sigma \times \sqrt{T}$$

In the above equations S is the stock price, K is the strike price, r is the risk free interest rate, T is the time to expiry of the option, σ and is the volatility. Φ is the standard normal cumulative distribution function.

The Base Price option is the call price of the most recent stock price. The Unit Price is calculated by subtracting the Base Price from the simulated Call Prices. By multiplying the Unit Price with the number of stocks held of a particular stock option we determine the profit and loss (P&L).

Step 7: VAR

To calculate VAR, the P&L values are calculated for each of the options on each of the Monte-Carlo scenarios. We add the P&L values for each option across a single scenario to get an aggregate scenario P&L value. In essence, what we've done is simulated the P&L for a portfolio 1000 times. It is from these simulated P&L values that we derive the VAR value.

Note that the sum of the individual security VAR numbers is greater than the aggregate portfolio VAR number.

5.5 <u>Programming Model:</u>

This section explains the implementation of the above mentioned Monte Carlo method.

The program retrieves the historical stock data from the database called "historical" via a ".txt" file, it then performs "preliminary calculations" (step 1 through step 5), then the Black-Scholes-Merton equations on the values obtained from the "preliminary calculations". Once the profit and loss (P&L) of each stock is calculated, the VAR is calculated.

The code has been divided in to three parts, namely:

- Preliminary Calculations
- Option Pricing
- Value At Risk estimation

5.6 Preliminary Calculations

Once the program reads the input data and finds no errors, it performs the initial calculations.

5.6.1 Calculate the T historical returns from the T historical prices of each of the n option

The historical returns are calculated by employing the formula:

$$r_t = \left[{(p_t - p_{(t-1)})}/{p_{(t-1)}} \right] \times 100$$

Where p_t is price at time t, $p_{(t-1)}$ is the price at time (t-1), and the return between the two periods is r_t.

5.6.2 Determine the $n \times n$ Variance- Covariance Matrix from the historical returns

The $n \times n$ Variance-Covariance Matrix is calculated by implementing the Variance-Covariance formula as described in step 3.

5.6.3 Calculate the Volatility of each stock option from the $n \times n$ Variance-Covariance Matrix

The volatility of each of the stock option is calculated by determining the square root of the diagonal elements of the $n \times n$ Variance-Covariance Matrix.

$$\sigma_i = \sqrt{diag(\Sigma)}$$

5.6.4 Determine the $n \times n$ Cholesky Decomposition from the Variance-Covariance Matrix

The symmetric $n \times n$ Variance-Covariance Matrix Σ, it can be decomposed in to a $n \times n$ diagonal Matrix D, a $n \times n$ lower triangular matrix L and an $n \times n$ upper triangular matrix L^Tsuch that:

$$\Sigma = LDL^T$$

Then the Cholesky Decomposition Matrix [4] is:

$$Chol = \sqrt{D}L^T$$

5.6.5 Generate m Normally Distributed Random Numbers for each stock option

The implementation is based on the polar form of the Boxed-Muller transformation [11]. The algorithm generates Gaussian pseudo-random numbers given a source of uniform pseudo-random numbers.

5.6.6 Calculate m Simulated Returns from the m Normally Distributed Random Numbers

The simulated returns are calculated by multiplying the $n \times n$ Normally Distributed Random Number Matrix with the transpose of the Cholesky Matrix. This generates a $m \times n$ matrix of simulated returns. Where, each column contains the simulated returns of a stock option.

5.6.7 Calculate m Simulated Prices from the m Simulated Returns

The simulated prices are generated by employing the formula explained in step 5.

5.7 Option Pricing

5.7.1 Based on the Monte Carlo Method the program execute the Black-Scholes equations. Details of the Black-Scholes method have been discussed in the step 6.

5.8 Value at Risk Estimation

The following steps are followed to calculate the Value at Risk:

5.8.1 Based on the desired confidence level (percentile), the value at risk is calculated, we have set the default value to 95%:

5.8.1.1 First the rank for the given percentile is calculated using the formula [12]:

$$rank = \frac{P}{100} \times (a + 1)$$

Where P, the confidence level is provide by the user and a are the number of simulated Profit and Loss values.

5.8.1.2 The Master node then chooses the smallest number pertaining to $rank$ after bubble sorting the Profit and Loss values. The value pertaining to the $rank$ is the VAR.

5.9 Implementation:

The above mentioned code works well and predicts the simulated stock price mentioned in step 5 and the Value-at-Risk in step 7.
Using this program the following three values are placed in the database:

1. A predicted price of each stock option
2. Value-At-Risk of each stock option
3. The programs opinion on whether to “Buy”, “Sell” or “Hold” the stock option.

Given the fact that we would generate several simulated prices for a single stock option, we use a random generator to pick one stock value for each option. It is this random stock value for each option that is present in the database. The Value at Risk for each stock option is calculated as specified in step 7. While the systems opinion on whether to buy, sell or hold is calculated by comparing the predicted price with the historical prices.

The user will retrieve the data pertaining to the program directly from the database and will have no interaction with program what so ever.

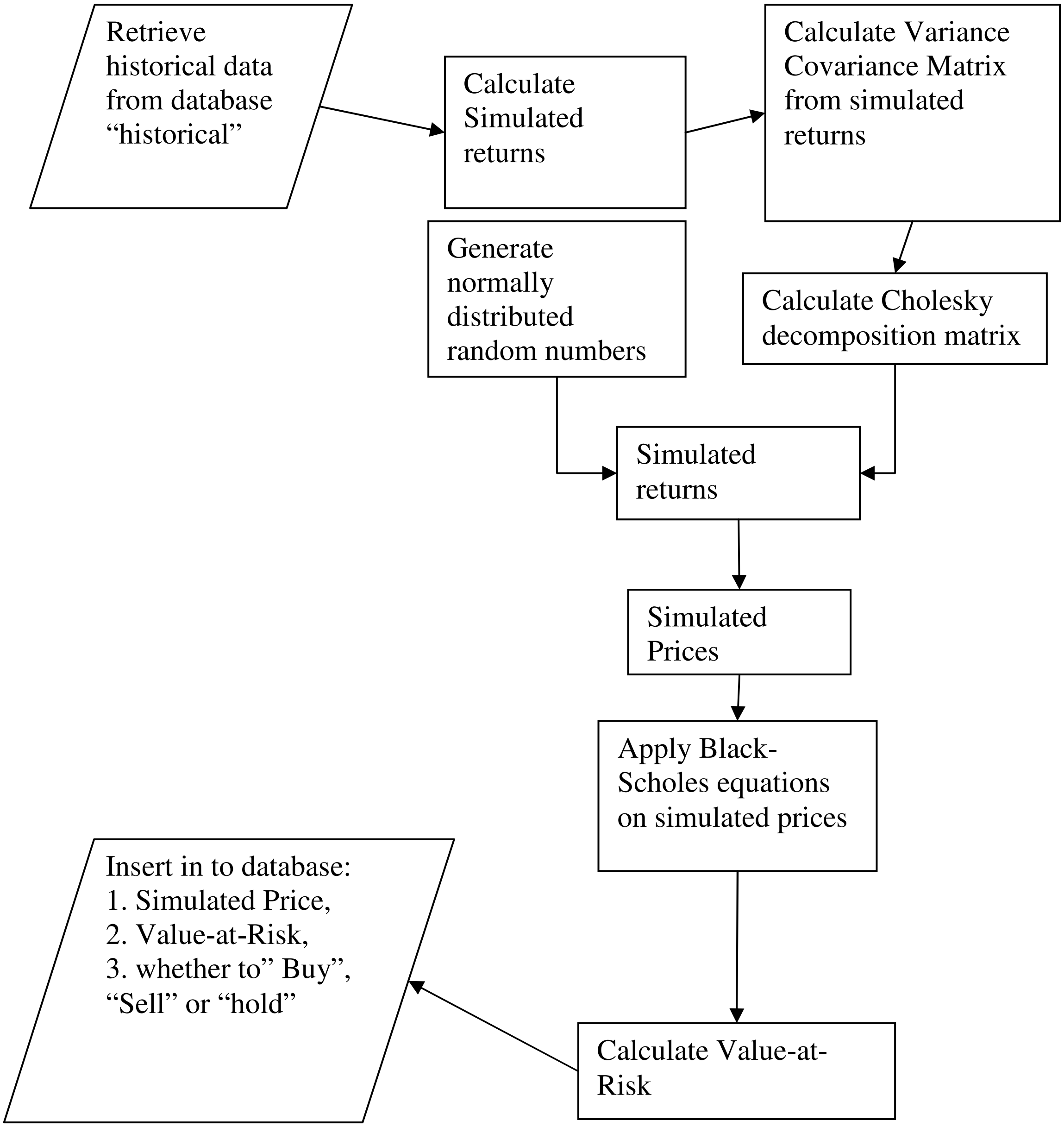

Retrieve historical data from database “historical”
Calculate Simulated returns
Calculate Variance Covariance Matrix from simulated returns
Generate normally distributed random numbers
Calculate Cholesky decomposition matrix
Simulated returns
Simulated Prices
Apply Black-Scholes equations on simulated prices
Insert in to database:
1. Simulated Price,
2. Value-at-Risk,
3. whether to” Buy”, “Sell” or “hold”
Calculate Value-at-Risk

Fig5.2: Flow chart describing the program

The class diagram displayed below has been derived from the explanation given above. These classes describes the methods used and their attributes.

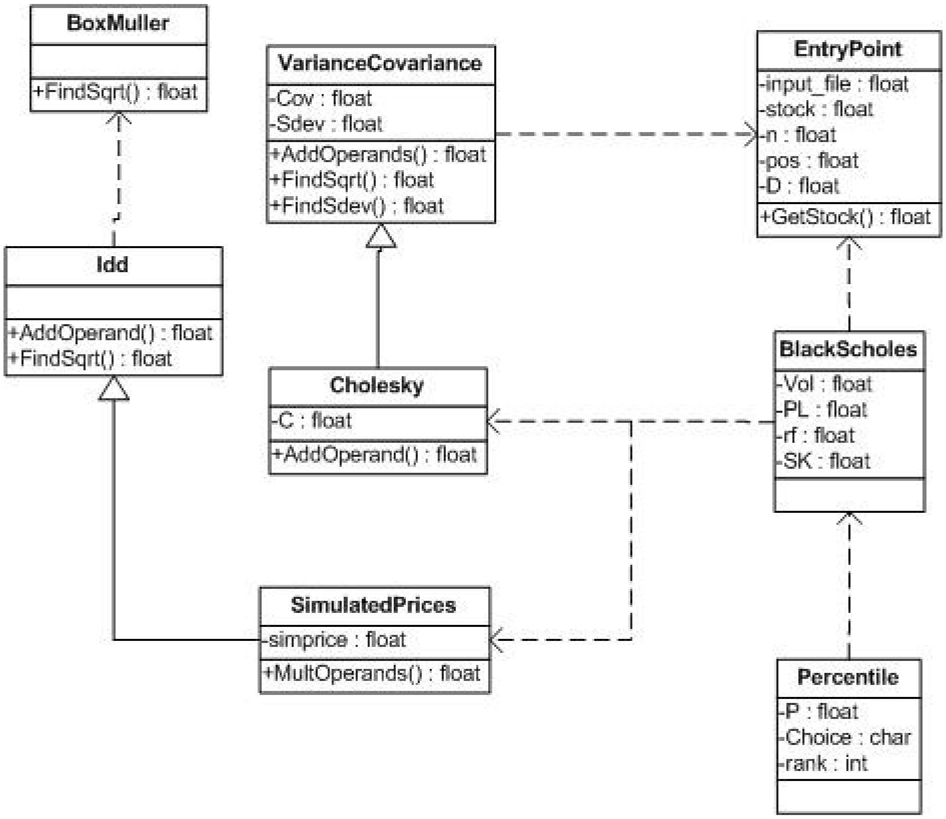


Fig.5.3: Class Diagram for the monte carlo predictor

6. Data Base Infrastructure

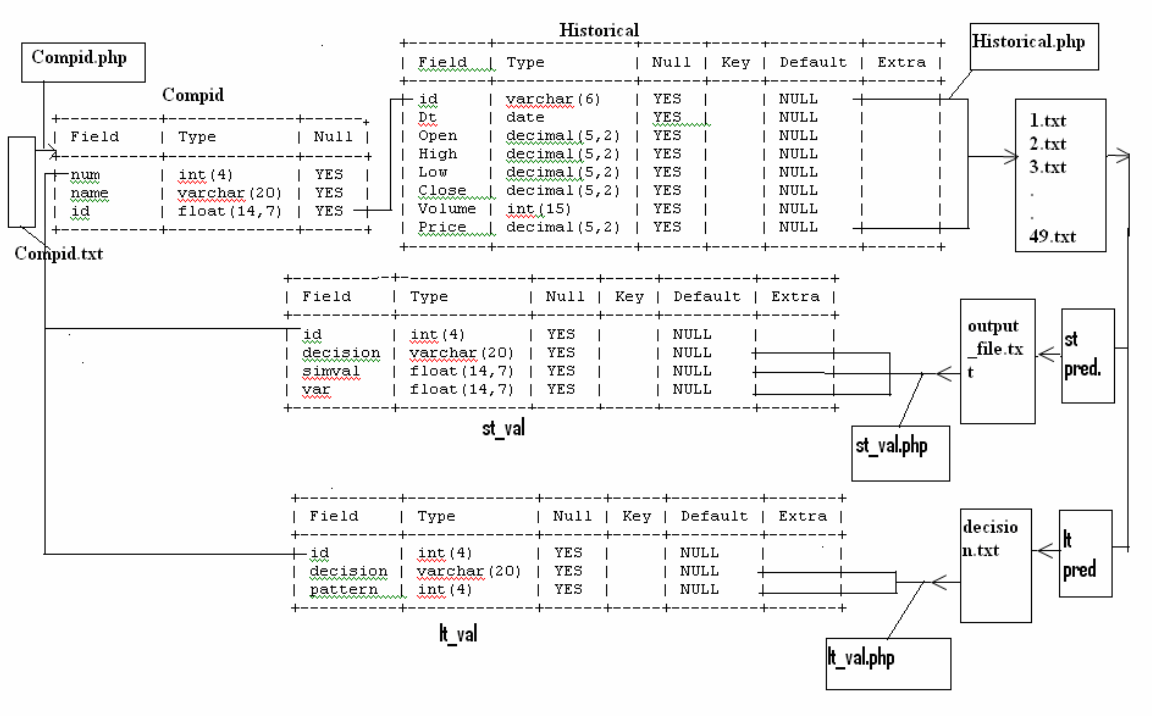
6.1 Data Collection

We used Yahoo! Finance API to collect our required data for this project. And to collect and periodically refresh those data to our database we developed a php program (previously it was a python code, but since we require historical data we made few changes and wrote PHP code). There is one PHP script running in the server continuously to update the previous "one year" stock data. The data is collected every twenty seconds as it is required for the short term predictor to refresh stock data and use updated data. As described above Monte Carlo predictor requires approximately three minutes to run the program and few more seconds to rewrite the result data base table, refreshing the actual stock price after thirty seconds prevents overlapping of any sort. We used only forty eight companies and not some two hundred companies because our resource is limited and our data base can not handle such magnitude of data.

To create the databases, we initially write a text file with the names of the companies and their respective Stock Exchange Symbol(for Example: Google,GOOG). This file is then read into the database using a PHP script(compid.php).During this process, each company is assigned a unique number which is used for referencing in the rest of the tables.

Once this table(compid) is populated, the id(for example: GOOG) is used as the Key to access the URL which provides the values(Date, Open, High, Low, Close, Volume, Price). Since our Predictors require values in the past one year, we manipulate the date function to explicitly get the values. These values are initially written into a csv file from which we extract the required values and then the table(Historical) is populated which is done using a PHP script(historical.php).

Once Historical is populated, it creates .txt files for each company with the unique number assigned to them in table (compid).
Whne the predictors finish their prediction, they return .txt files with the respective values. The values are then extracted and then populated in the table(st_val) & (lt_val). The PHP scripts used to do these functions are automated to run in the background at properly spaced intervals, so that the tables maintain the latest values.



Field	Type	Null
num	int(4)	YES
name	varchar(20)	YES
id	float(14,7)	YES

Field	Type	Null	Key	Default	Extra
id	varchar(6)	YES		NULL	
Dt	date	YES		NULL	
Open	decimal(5,2)	YES		NULL	
High	decimal(5,2)	YES		NULL	
Low	decimal(5,2)	YES		NULL	
Close	decimal(5,2)	YES		NULL	
Volume	int(15)	YES		NULL	
Price	decimal(5,2)	YES		NULL	

Field	Type	Null	Key	Default	Extra
id	int(4)	YES		NULL	
decision	varchar(20)	YES		NULL	
simval	float(14,7)	YES		NULL	
var	float(14,7)	YES		NULL	

Field	Type	Null	Key	Default	Extra
id	int(4)	YES		NULL	
decision	varchar(20)	YES		NULL	
pattern	int(4)	YES		NULL	

Fig 6.1 illustrates the interactions of different tables and modules. Note: var=value at risk.

6.2 Schemas and table entries

We are tracking forty eight companies (as one company data we were retrieving was corrupt), we randomly choose these famous companies. Companies that we are tracking according to there symbols are:

"AAPL","CSCO","MSFT","INTC","NOK","HPQ","ORCL","DELL","CAJ","TXN", "EMC","MOT","AMD","JNJ","PFE","GSK","NVS","MRK","AZN","NOC","AMGN", "SNY","ABT","WYE","DNA","ACL","C","BAC","HBC","BRK-A","AIG","JPM", "AZ","WB","LFC","GS","BCS","AXA","XOM","GME","CVX","BP","OXY","IMO", "CHL","T","VOD","GOOG","WMT","VZ".

As discussed earlier there are three tables in our database one each for receiving the predictor results(table name of Monte Carlo predictors result is st_val and that of SVM predictor is lt_val) and one for historical stock data(name of the table is historical) which includes company id, date, opening price, closing price, day's low, day's high, volume and current price which are required for our predictors . While the Monte Carlo result table contains four columns-company id, decision (simulated decision), simval(simulated value), var(value at risk) while the SVM predictor result table contains three column company id, decision and pattern. Due to long table entries we are not able show it on the report.

Table 6.1 st_val(Monte Carlo predictor) schema:

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(4)      | YES  |     | NULL    |       |
| decision | varchar(20) | YES  |     | NULL    |       |
| simval   | float(14,7) | YES  |     | NULL    |       |
| var      | float(14,7) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
4 rows in set (0.02 sec)
```

Table 6.2 st_val table entries(Monte Carlo predictor's simulated results):

```
+------+----------+-------------+----------------+
| id   | decision | simval      | var            |
+------+----------+-------------+----------------+
|    0 | Sell     | 186.3476562 |  -3368.2770996 |
|    1 | Buy      |  26.1063957 |   -300.7302246 |
|    2 | Sell     |  29.7465515 |   -219.5667419 |
|    3 | Buy      |  23.0005646 |   -289.3081970 |
|    4 | Sell     |  29.8489380 |   -296.9724731 |
|    5 | Sell     |  48.5851173 |   -470.8375854 |
|    6 | Sell     |  22.2794304 |   -239.8322144 |
|    7 | Sell     |  19.6728134 |   -203.8833618 |
|    8 | Buy      |  50.9559212 |   -622.1882935 |
|    9 | Buy      |  29.1142216 |   -235.8962860 |
|   10 | Sell     |  16.1348572 |   -258.2118835 |
|   11 | Hold     |   9.9283295 |   -110.0219116 |
|   12 | Buy      |   6.4473009 |   -101.3148117 |
|   13 | Buy      |  67.5549545 |   -410.6645813 |
|   14 | Hold     |  20.4657650 |   -130.3724213 |
|   15 | Buy      |  44.4124451 |   -436.3081970 |
|   16 | Sell     |  51.0786095 |   -418.2741089 |
|   17 | Buy      |  37.7119141 |   -346.6322632 |
|   18 | Sell     |  42.7416687 |   -523.9976196 |
|   19 | Sell     |  75.8909302 |   -423.2734070 |
|   20 | Buy      |  41.9588547 |   -309.7172852 |
|   21 | Sell     |  40.1930618 |   -319.0434265 |
|   22 | Buy      |  51.8456421 |   -526.9990845 |
|   23 | Sell     |  46.5568390 |   -370.3628235 |
|   24 | Sell     |  69.6842957 |   -228.2865295 |
|   25 | Sell     | 163.5313873 |  -1446.4038086 |
|   26 | Buy      |  24.6074982 |   -430.7090454 |
|   27 | Buy      |  38.1520844 |   -137.8429871 |
|   28 | Hold     |  87.2881775 |   -801.4334717 |
|   29 | Buy      |  46.2998581 |   -601.4199829 |
|   30 | Sell     |  48.6901398 |   -590.4094849 |
|   31 | Hold     | 126.6664581 |   -248.9327087 |
|   32 | Sell     |  31.3698235 |   -551.6023560 |
|   33 | Buy      |  64.4160767 |  -1538.9218750 |
|   34 | Sell     | 203.3425140 |  -3324.4638672 |
|   35 | Sell     |  39.1456413 |   -378.8312683 |
|   36 | Hold     |  37.7010078 |   -506.2254944 |
|   37 | Sell     |  90.6620102 |   -601.0397339 |
|   38 | Sell     |  53.3684540 |   -437.6981506 |
|   39 | Hold     |  95.6339035 |   -957.9890137 |
|   40 | Sell     |  72.8459778 |   -837.0157471 |
|   41 | Buy      |  83.9974976 |  -1086.3925781 |
|   42 | Sell     |  56.0843735 |   -352.5381775 |
|   43 | Hold     |  86.6920395 |  -1305.9456787 |
|   44 | Sell     |  40.1907158 |   -411.8206177 |
|   45 | Buy      |  31.5982895 |   -275.8382874 |
|   46 | Sell     | 602.6482544 | -10767.5615234 |
|   47 | Sell     |  57.0924988 |   -459.1134033 |
|   48 | Sell     |  39.4091988 |   -392.9883118 |
+------+----------+-------------+----------------+
49 rows in set (0.01 sec)
```

Table 6.3 Lt_val(SVM pattern Predictor) schema:

```
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| id       | int(4)      | YES  |     | NULL    |       |
| decision | varchar(20) | YES  |     | NULL    |       |
| pattern  | int(4)      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

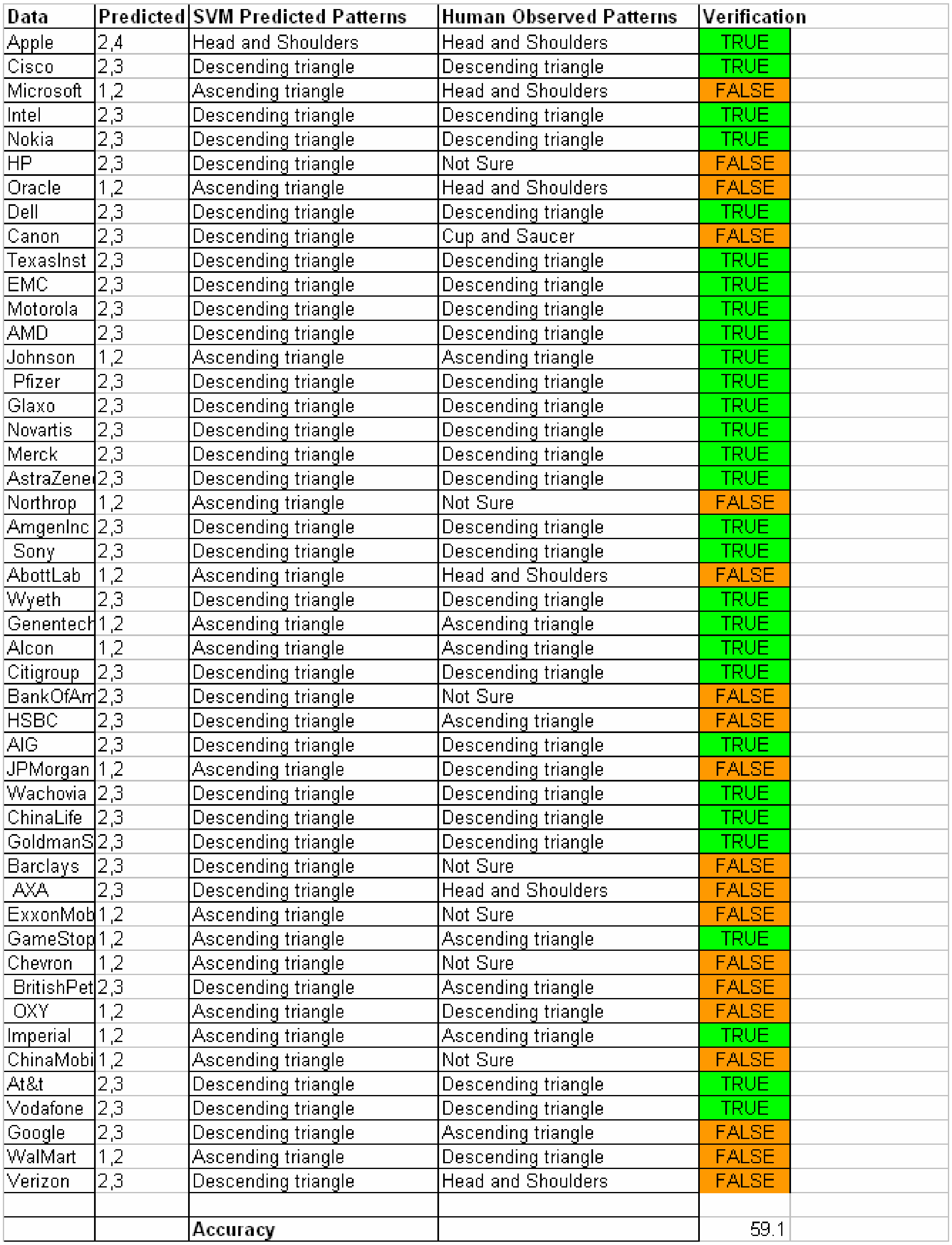
Table 6.4 Historical(obtained from Yahoo! Finance) schema:

```
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| id     | varchar(6)   | YES  |     | NULL    |       |
| Dt     | date         | YES  |     | NULL    |       |
| Open   | decimal(5,2) | YES  |     | NULL    |       |
| High   | decimal(5,2) | YES  |     | NULL    |       |
| Low    | decimal(5,2) | YES  |     | NULL    |       |
| Close  | decimal(5,2) | YES  |     | NULL    |       |
| Volume | int(15)      | YES  |     | NULL    |       |
| Price  | decimal(5,2) | YES  |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
8 rows in set (0.05 sec)
```

7. Results and Evaluations:

We ran the SVM predictor algorithm on real time data from Yahoo finance for 48 companies. Our predictor successfully identified 59.1% of real time data patterns and was trained for four patterns. The predictions of our SVM predictor were verified against human labeled data patterns. Below are the results and actual data patterns.

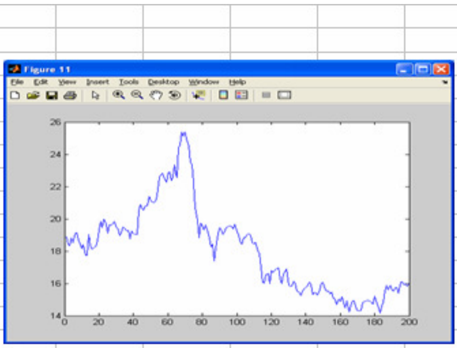
SVM Algorithm Simulation Results:

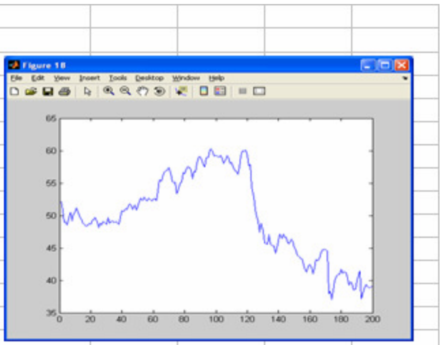


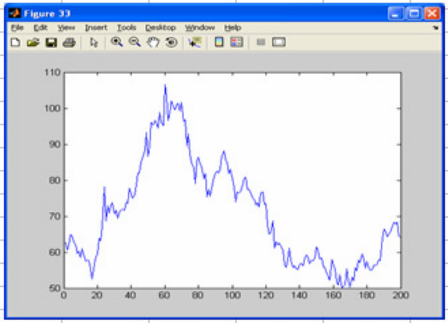
Data	Predicted	SVM Predicted Patterns	Human Observed Patterns	Verification
Apple	2,4	Head and Shoulders	Head and Shoulders	TRUE
Cisco	2,3	Descending triangle	Descending triangle	TRUE
Microsoft	1,2	Ascending triangle	Head and Shoulders	FALSE
Intel	2,3	Descending triangle	Descending triangle	TRUE
Nokia	2,3	Descending triangle	Descending triangle	TRUE
HP	2,3	Descending triangle	Not Sure	FALSE
Oracle	1,2	Ascending triangle	Head and Shoulders	FALSE
Dell	2,3	Descending triangle	Descending triangle	TRUE
Canon	2,3	Descending triangle	Cup and Saucer	FALSE
TexasInst	2,3	Descending triangle	Descending triangle	TRUE
EMC	2,3	Descending triangle	Descending triangle	TRUE
Motorola	2,3	Descending triangle	Descending triangle	TRUE
AMD	2,3	Descending triangle	Descending triangle	TRUE
Johnson	1,2	Ascending triangle	Ascending triangle	TRUE
Pfizer	2,3	Descending triangle	Descending triangle	TRUE
Glaxo	2,3	Descending triangle	Descending triangle	TRUE
Novartis	2,3	Descending triangle	Descending triangle	TRUE
Merck	2,3	Descending triangle	Descending triangle	TRUE
AstraZene	2,3	Descending triangle	Descending triangle	TRUE
Northrop	1,2	Ascending triangle	Not Sure	FALSE
AmgenInc	2,3	Descending triangle	Descending triangle	TRUE
Sony	2,3	Descending triangle	Descending triangle	TRUE
AbottLab	1,2	Ascending triangle	Head and Shoulders	FALSE
Wyeth	2,3	Descending triangle	Descending triangle	TRUE
Genentech	1,2	Ascending triangle	Ascending triangle	TRUE
Alcon	1,2	Ascending triangle	Ascending triangle	TRUE
Citigroup	2,3	Descending triangle	Descending triangle	TRUE
BankOfAm	2,3	Descending triangle	Not Sure	FALSE
HSBC	2,3	Descending triangle	Ascending triangle	FALSE
AIG	2,3	Descending triangle	Descending triangle	TRUE
JPMorgan	1,2	Ascending triangle	Descending triangle	FALSE
Wachovia	2,3	Descending triangle	Descending triangle	TRUE
ChinaLife	2,3	Descending triangle	Descending triangle	TRUE
GoldmanS	2,3	Descending triangle	Descending triangle	TRUE
Barclays	2,3	Descending triangle	Not Sure	FALSE
AXA	2,3	Descending triangle	Head and Shoulders	FALSE
ExxonMob	1,2	Ascending triangle	Not Sure	FALSE
GameStop	1,2	Ascending triangle	Ascending triangle	TRUE
Chevron	1,2	Ascending triangle	Not Sure	FALSE
BritishPet	2,3	Descending triangle	Ascending triangle	FALSE
OXY	1,2	Ascending triangle	Descending triangle	FALSE
Imperial	1,2	Ascending triangle	Ascending triangle	TRUE
ChinaMobi	1,2	Ascending triangle	Not Sure	FALSE
At&t	2,3	Descending triangle	Descending triangle	TRUE
Vodafone	2,3	Descending triangle	Descending triangle	TRUE
Google	2,3	Descending triangle	Ascending triangle	FALSE
WalMart	1,2	Ascending triangle	Descending triangle	FALSE
Verizon	2,3	Descending triangle	Head and Shoulders	FALSE
		Accuracy		59.1

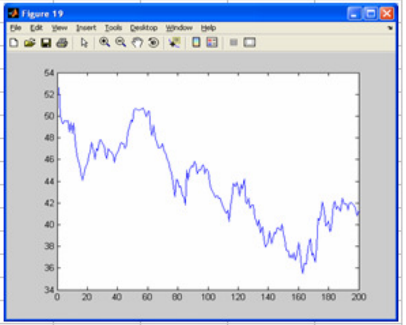
Patterns

Descending Triangle

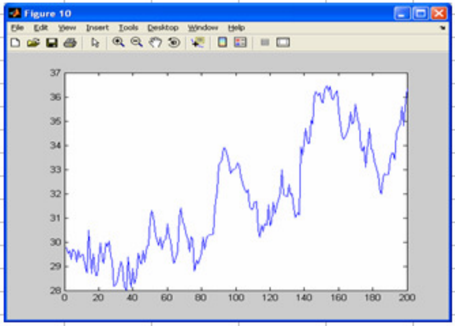




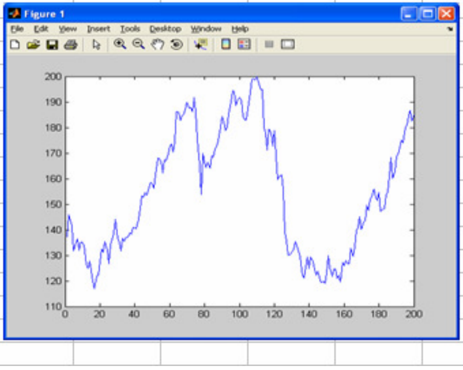




Ascending Triangle



Head and Shoulder

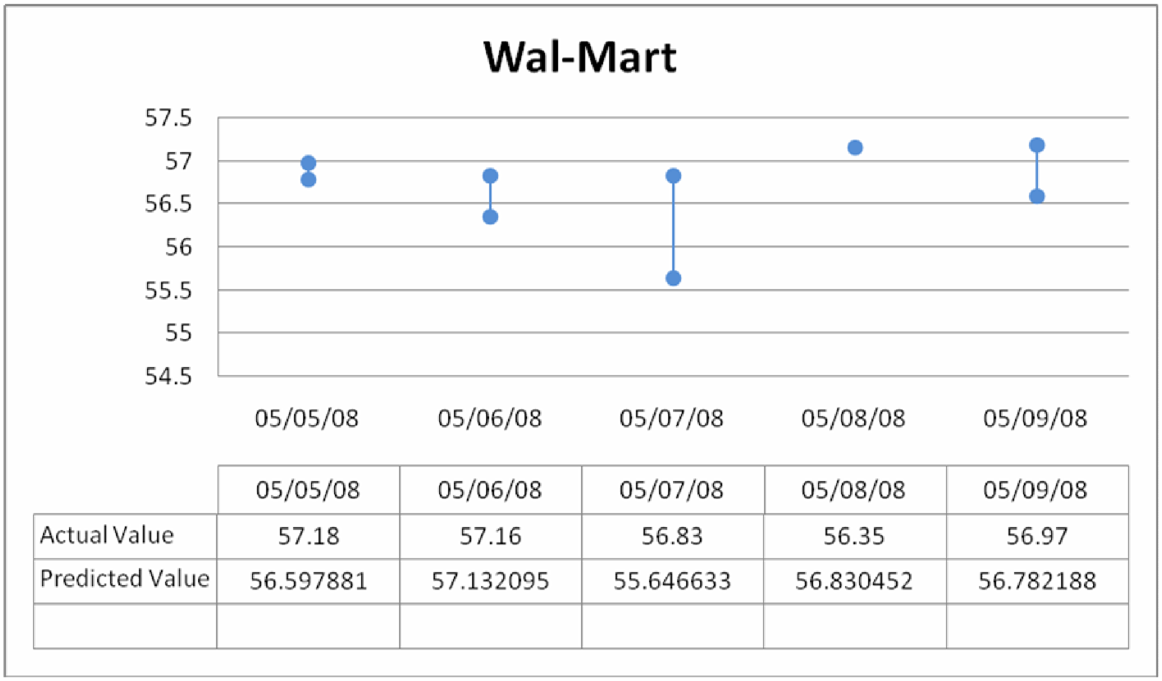


Monte Carlo Algorithm Simulation Results:

We have explained the working of our algorithms in the above sections. In this section we evaluate the output of our algorithm. Here we have compared the the predicted values (in dollars) after running our program with the actual values (in dollars) of 4 companies for the one weeks time preiod (from May 2, 2008 to May 9, 2008). In order to obtain these results we used one years historical stock data from May 2, 2007 to t. Where t is the day before the predicted value was calculated. For example, to predict the stock value for May 6, 2008 we used historical data from May 2, 2007 to May 5, 2008 while to predict the value for May 9, 2008 we used historical data from May 2, 2007 to May 8, 2008. The results obtained have been plotted in the form of error graphs.

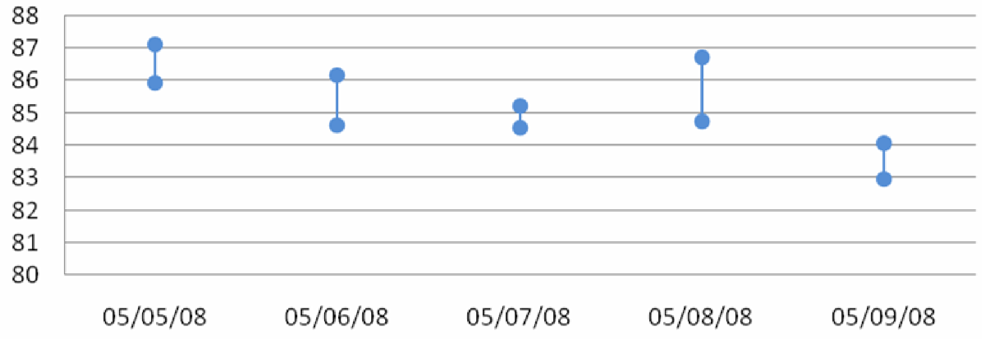
It can be seen from the error graphs that the error between the actual and predicted values are very small. The largest error between the predicted and actual values is observed in Google on May 8th, 2008, the error being 10.667861 (in dollars). While the smallest error is observed in Johnson & Johnson on May 9th, 2008 the error being 0.009654 (in dollars). In fact, the error in Johnson & Johnson is the least with almost similar values between the predicted and actual values.

The graphs plotted below are self explanatory. They display the error between the predicted and actual values for a week for 4 companies namely, Wal-Mart, Boeing, Johnson & Johnson and Google. They also display a table that contains the actual and predicted values.



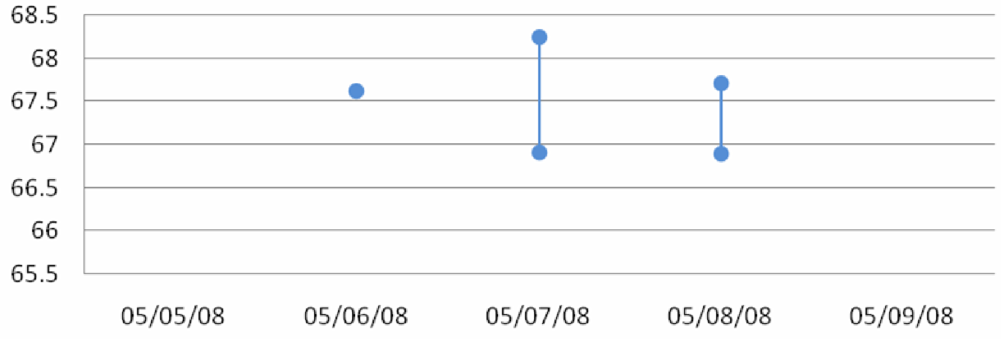
	05/05/08	05/06/08	05/07/08	05/08/08	05/09/08
Actual Value	57.18	57.16	56.83	56.35	56.97
Predicted Value	56.597881	57.132095	55.646633	56.830452	56.782188

Boeing

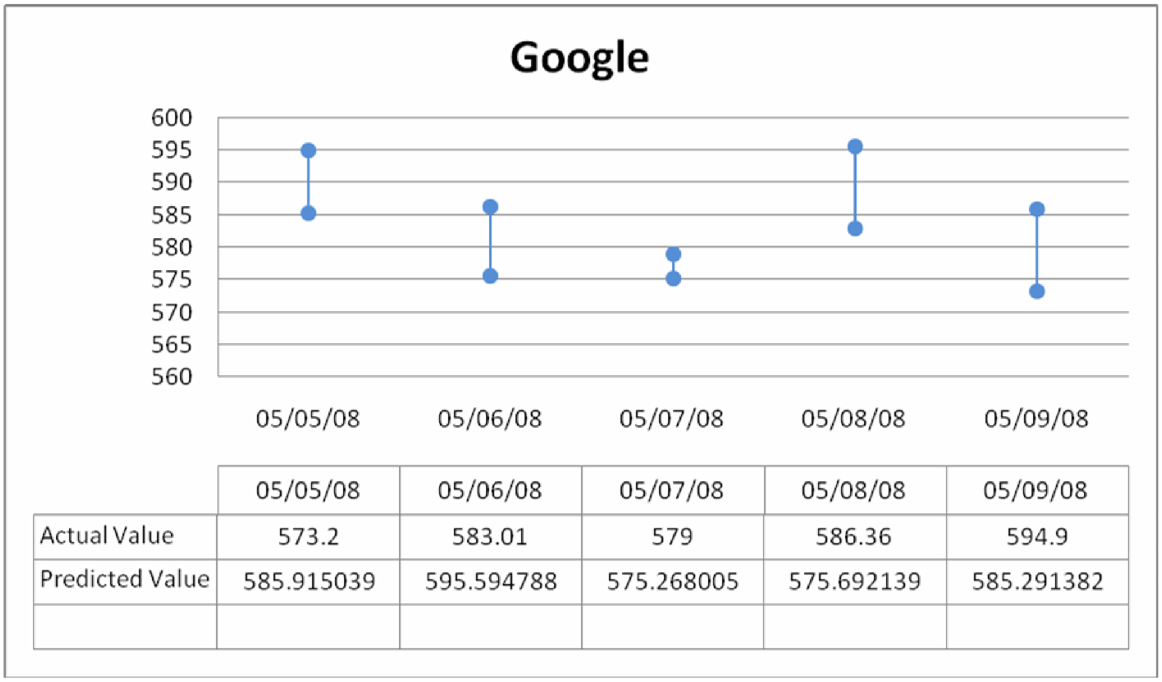


	05/05/08	05/06/08	05/07/08	05/08/08	05/09/08
Actual Value	84.06	84.76	84.55	86.15	85.92
Predicted Value	82.97261	86.710884	85.199425	84.61097	87.104111

Johnson & Johnson



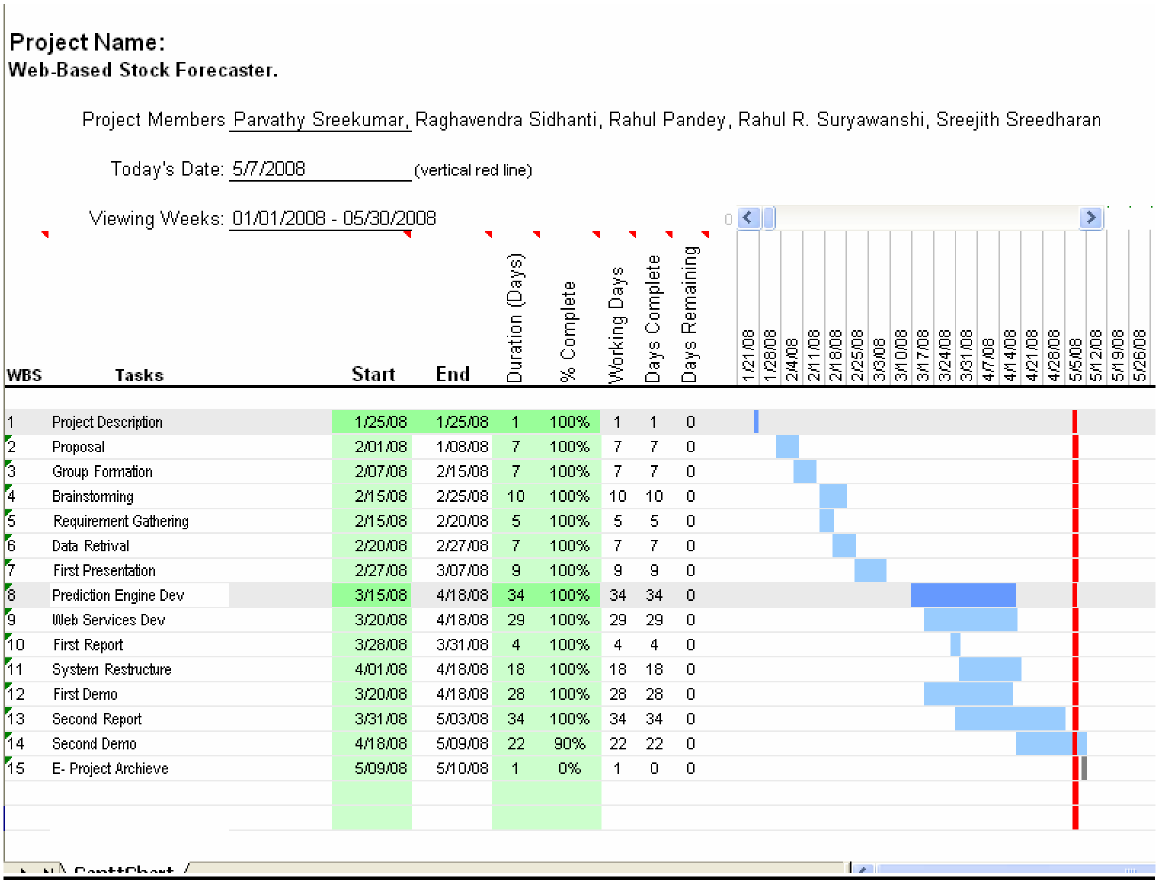
	05/05/08	05/06/08	05/07/08	05/08/08	05/09/08
Actual Value	66.55	66.9	66.91	67.58	67.9
Predicted Value	66.559654	67.702881	68.243744	67.627487	67.902451



	05/05/08	05/06/08	05/07/08	05/08/08	05/09/08
Actual Value	573.2	583.01	579	586.36	594.9
Predicted Value	585.915039	595.594788	575.268005	575.692139	585.291382

8. Plan Of Work

We were able to distribute our work equally and completed our work in time. The following is the Gantt Chart description of our plan of work from the start of the semester to the end of semester.



Project Name:
Web-Based Stock Forecaster.

Project Members Parvathy Sreekumar, Raghavendra Sidhanti, Rahul Pandey, Rahul R. Suryawanshi, Sreejith Sreedharan

Today's Date: 5/7/2008 (vertical red line)

Viewing Weeks: 01/01/2008 - 05/30/2008

WBS	Tasks	Start	End	Duration (Days)	% Complete	Working Days	Days Complete	Days Remaining
1	Project Description	1/25/08	1/25/08	1	100%	1	1	0
2	Proposal	2/01/08	1/08/08	7	100%	7	7	0
3	Group Formation	2/07/08	2/15/08	7	100%	7	7	0
4	Brainstorming	2/15/08	2/25/08	10	100%	10	10	0
5	Requirement Gathering	2/15/08	2/20/08	5	100%	5	5	0
6	Data Retrival	2/20/08	2/27/08	7	100%	7	7	0
7	First Presentation	2/27/08	3/07/08	9	100%	9	9	0
8	Prediction Engine Dev	3/15/08	4/18/08	34	100%	34	34	0
9	Web Services Dev	3/20/08	4/18/08	29	100%	29	29	0
10	First Report	3/28/08	3/31/08	4	100%	4	4	0
11	System Restructure	4/01/08	4/18/08	18	100%	18	18	0
12	First Demo	3/20/08	4/18/08	28	100%	28	28	0
13	Second Report	3/31/08	5/03/08	34	100%	34	34	0
14	Second Demo	4/18/08	5/09/08	22	90%	22	22	0
15	E- Project Archieve	5/09/08	5/10/08	1	0%	1	0	0

Fig 8.1 Detail plan of action for web based stock forecaster.

9. Reference:

[1] Pattern Recognition and Machine Learning – Christopher M. Bishop http://research.microsoft.com/~cmbishop/prml/

[2] A Tutorial on Support Vector Machines for Pattern Recognition http://research.microsoft.com/~cburges/papers/SVMTutorial.pdf

[3] Neural Network Theory - A Short Tutorial http://documents.wolfram.com/applications/neuralnetworks/index2.html

[4] Comparison of Support Vector Machine and Artificial Neural Network Systems for Drug/Nondrug Classification - Evgeny Byvatov, Uli Fechner, Jens Sadowski, and Gisbert Schneider http://pubs.acs.org/cgi-bin/abstract.cgi/jcisd8/2003/43/i06/abs/ci0341161.html

[5] Learning Capability: Classical RBF Network vs. SVM with Gaussian Kernel - Rameswar Debnath and Haruhisa Takahashi http://www.springerlink.com/content/feg023aac516wwje/

[6] A Case Study on Support Vector Machines Versus Artificial Neural Networks - Wen-Chyi Lin, Chung-Cheng Institute of Technology, 1998.

[7] Support Vector Machines Versus Neural Networks - Huzefa Rangwala and Jayashree Sadagopan, Mar 2004.

[8] The SPIDER: Machine Learning Toolbox in Matlab http://www.kyb.tuebingen.mpg.de/bs/people/spider/

[9] Chris Marty. Value-at-Risk Project background Information document. Bloomberg L.P.

[10]Gilbert Strang book on: Linear Algebra And Its Applications.

[11] Generating Gaussian Random Numbers. http://www.taygeta.com/random/gaussian/.

[12] David Lane. Percentiles. http://www.cnx.org/content/m10805/latest/.

[13]Investopedia. http://www.investopedia.com/articles/04/092904.asp,

http://www.investopedia.com/articles/04/101304.asp

[14]Wikipedia. http://en.wikipedia.org/wiki/Black-Scholes/

[15] Ascending Triangle pattern
http://www.trending123.com/patterns/ascending_triangle.html

[16] Cup and Saucer pattern
http://www.trending123.com/patterns/cup_and_handle.html

[17] Descending Triangle http://www.trending123.com/patterns/descending_triangle.html

[18] Technical analysis http://www.investopedia.com/terms/t/technicalanalysis.asp

[19] Head and Shoulders pattern
http://www.trending123.com/patterns/head_and_shoulders.html

[20] Support vector machine for regression and applications to financial forecasting - Trafalis, T.B.; Ince, H., Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Volume 6, Issue , 2000 Page(s):348 - 353 vol.6

[21] Machine Learning Techniques and Use of Event Information for Stock Market Prediction: A Survey and Evaluation - Yoo, P.D.; Kim, M.H.; Jan, T. Computational Intelligence for Modelling, Control and Automation, 2005

[22]Web Services and Other Distributed Technologies http://msdn.microsoft.com/en-us/webservices/default.aspx

[23]Software Engineering,http://software-ece.rutgers.edu/~group801/MC_VaR.pdf

[24]Creating web services in .Net http://www.west-wind.com/presentations/dotnetwebservices/DotNetWebServices.asp

[25] Risk Management: An introduction to Value at Risk. Thomas J. Linsmeier and Neil

D. Pearson. University of Illinois at Urbana-Champaign. An

www.exinfm.com/training/pdfiles/valueatrisk.pdf

[26] Implementing Value at Risk. Wharton School at University of Pennsylvania.

finance.wharton.upenn.edu/~benninga/mma/MiER74.pdf